21世纪高等学校规划教材 | 软件工程





软件工程

任永昌 编著

清华大学出版社

21 世纪高等学校规划教材·软件工程

软件工程

任永昌　编著

清华大学出版社
北 京

内 容 简 介

本书从软件工程的基本原理出发，以适应本科专业的教学和实践为宗旨，在充分吸收国内外软件工程最新研究成果精华的基础上，依据作者多年软件工程领域的教学和科研经验，结合国内软件项目开发与维护的特点编写而成。

本书按技术篇、管理篇、实验篇的顺序编写。技术篇共9章，分别讲述软件工程概述、软件开发过程模型、可行性研究、需求分析、概要设计、详细设计、软件实现、软件测试、软件维护；管理篇共6章，分别讲述进度计划管理、质量管理、成本管理、配置管理、文档管理、人力资源管理；实验篇设计了10个与软件开发过程密切相关的文档书写。

本书可作为高等学校计算机及相关专业本科生"软件工程"课程的教材，也可作为希望了解软件工程思想、技术以及软件项目管理方法的各类读者的参考书。

图书在版编目（CIP）数据

软件工程/任永昌编著.--北京：清华大学出版社，2012.7
（21世纪高等学校规划教材·软件工程）
ISBN 978-7-302-28551-9

Ⅰ.①软…　Ⅱ.①任…　Ⅲ.①软件工程－高等学校－教材　Ⅳ.①TP311.5

中国版本图书馆CIP数据核字(2012)第067064号

责任编辑：郑寅堃　王冰飞
封面设计：
责任校对：白　蕾
责任印制：

出版发行：清华大学出版社
网　址：http://www.tup.com.cn，http://www.wqbook.com
地　址：北京清华大学学研大厦A座　**邮　编**：100084
社 总 机：010-62770175　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185mm×260mm　**印　张**：19.75　**字　数**：475千字
版　次：2012年7月第1版　**印　次**：2012年7月第1次印刷
印　数：1～　000
定　价：　.00元

产品编号：045915-01

编审委员会成员

（按地区排序）

浙江大学	吴朝晖　教授
	李善平　教授
扬州大学	李　云　教授
南京大学	骆　斌　教授
	黄　强　副教授
南京航空航天大学	黄志球　教授
	秦小麟　教授
南京理工大学	张功萱　教授
南京邮电学院	朱秀昌　教授
苏州大学	王宜怀　教授
	陈建明　副教授
江苏大学	鲍可进　教授
中国矿业大学	张　艳　教授
武汉大学	何炎祥　教授
华中科技大学	刘乐善　教授
中南财经政法大学	刘腾红　教授
华中师范大学	叶俊民　教授
	郑世珏　教授
	陈　利　教授
江汉大学	颜　彬　教授
国防科技大学	赵克佳　教授
	邹北骥　教授
中南大学	刘卫国　教授
湖南大学	林亚平　教授
西安交通大学	沈钧毅　教授
	齐　勇　教授
长安大学	巨永锋　教授
哈尔滨工业大学	郭茂祖　教授
吉林大学	徐一平　教授
	毕　强　教授
山东大学	孟祥旭　教授
	郝兴伟　教授
厦门大学	冯少荣　教授
厦门大学嘉庚学院	张思民　教授
云南大学	刘惟一　教授
电子科技大学	刘乃琦　教授
	罗　蕾　教授
成都理工大学	蔡　淮　教授
	于　春　副教授
西南交通大学	曾华燊　教授

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的"高等学校教学质量与教学改革工程精品课程建设工作"，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了"清华大学出版社教材编审委员会"(以下简称"编委会")，旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。"编委会"成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，"编委会"一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性(新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的"编委会"成员推荐)，经"编委会"认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

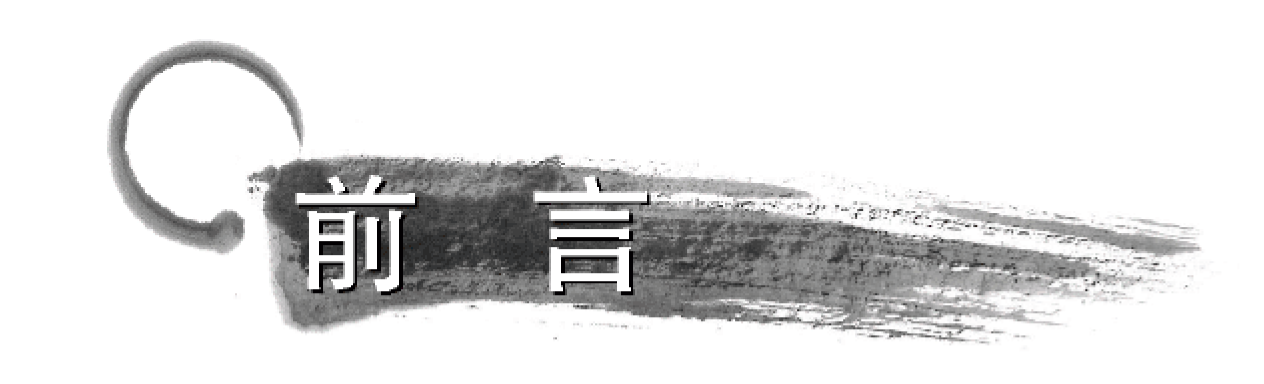
（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人：魏江江
E-mail：weijj@tup.tsinghua.edu.cn



软件产业是信息技术领域发展最快的产业，是增强综合国力的关键产业，是信息产业的核心和灵魂，软件产业的发展规模和水平已经成为衡量一个国家现代化程度和竞争力的重要标志。随着计算机技术的应用和普及，计算机程序日益复杂，软件开发出现的问题越来越难以解决，集中表现为对软件开发的成本和进度估算不准确、用户对已完成的软件不满意、软件可维护性差、软件质量不可靠、软件产品供不应求、软件产品价格昂贵、软件缺乏适当的文档资料等问题，解决这些问题的有效途径就是软件工程。

软件工程是研究和指导软件开发和维护的工程类学科，以计算机科学理论及其他信息技术理论为指导，采用工程化的概念、原理、规范、技术和方法进行软件工程项目的开发和维护，把经过实践证明正确的管理措施和当前能够得到的最好的技术方法结合起来，以较少的代价获取高质量的软件产品。通过 50 余年的努力，软件工程已逐步发展为一门成熟的专业学科。软件工程作为一门专业主干课，重点要求学生学习与软件开发和维护有关的四个方面的内容——过程与模型、方法与技术、工具与环境、标准与规范。进而通过课程实践培养学生运用软件工程基本原理解决实际问题，并从事复杂软件项目开发和维护的实践应用能力与创新能力，使他们成为当今信息社会和知识经济时代所需要的高素质计算机专业人才。

本书以软件工程的思想和方法为指导，充分吸收国内外软件工程最新研究成果，结合软件项目开发的实际情况编写。但软件项目有其独特性，开发的成功不仅依赖于成熟先进的方法和技术，更依赖于人的素质和技能。读者在借鉴书中思想的同时，应不断探索软件开发的新理论与新方法，提高软件项目的成功率。

本书分为技术篇、管理篇和实验篇，主要内容和结构如下：

技术篇共 9 章，按照软件生命周期过程讲述与软件项目开发和维护有关的技术问题。

第 1 章软件工程概述，主要讲述软件及其特征、软件危机及其表现和原因、软件工程的概念和原理、软件工程方法学、软件项目管理基础。

第 2 章软件开发过程模型，主要讲述软件生命周期、软件过程、软件开发过程、软件开发过程模型、软件开发过程模型选择。

第 3 章可行性研究，主要讲述可行性研究的含义与内容、可行性研究的阶段、成本/效益分析、方案选择与决策、可行性研究报告的描述方法。

第 4 章需求分析，主要讲述需求分析概述、需求分析过程、需求分析内容、需求分析方法、需求分析变更、需求分析验证、需求管理。

第 5 章概要设计，主要讲述概要设计的设计任务、设计原则、图形工具、设计方法、启发式设计策略，简要说明接口设计、概要设计与详细设计的衔接。

第 6 章详细设计，主要讲述设计任务、结构程序设计、表示工具(包括流程图、盒图、问题分析图、过程设计语言、IPO 图、判定树、判定表等)、面向数据结构的设计方法、程序复杂性度量。

第 7 章软件实现，主要讲述输入设计、输出设计、屏幕界面设计、程序设计语言、编程风格、软件调试、程序效率、程序安全性。

第 8 章软件测试，主要讲述软件的测试过程和原则、静态测试与动态测试、黑盒测试与白盒测试、测试用例设计技术、单元测试、集成测试、确认测试、系统测试。

第 9 章软件维护，主要讲述软件维护概述、软件维护过程模型、软件维护技术、软件维护过程、软件维护控制、软件维护组织管理、软件再工程。

管理篇共 6 章，讲述与软件工程过程密切相关的管理问题。

第 10 章进度计划管理，主要讲述软件项目进度计划概述、甘特图法和持续时间压缩法等进度计划编制方法、进度计划的表达形式及编制过程、进度计划控制的难点和手段。

第 11 章质量管理，主要讲述软件质量与软件质量管理、软件质量策划、软件质量计划、软件质量保证、软件质量控制、软件质量改进、软件评审、ISO 9000 质量管理体系。

第 12 章成本管理，主要讲述软件成本特点、软件成本构成、软件成本影响因素、软件资源计划、软件成本估算、软件成本预算、软件成本控制。

第 13 章配置管理，主要讲述配置管理需求分析、配置管理的作用、配置管理的相关概念、配置管理的组织、配置管理的主要活动。

第 14 章文档管理，主要讲述文档管理的概念、文档与软件规模、文档的分类与作用、文档编制要求、文档编制过程、文档编制相关问题。

第 15 章人力资源管理，主要讲述软件项目人力资源的特征、人力资源管理的主要内容、人员的组织与分工、人力资源计划、项目经理、团队建设。

实验篇设计了在软件工程过程中最重要的 10 个文档书写，分别是可行性分析报告、软件需求规格说明、软件结构设计说明、软件详细设计说明、软件测试报告、软件产品规格说明、软件开发计划、软件质量保证计划、软件配置管理计划、软件用户手册，通过文档书写，使学生掌握较强的软件开发的组织、管理、实施方法和技术，促进学生理解软件工程的理论知识、标准和规范，提高学生的综合能力。

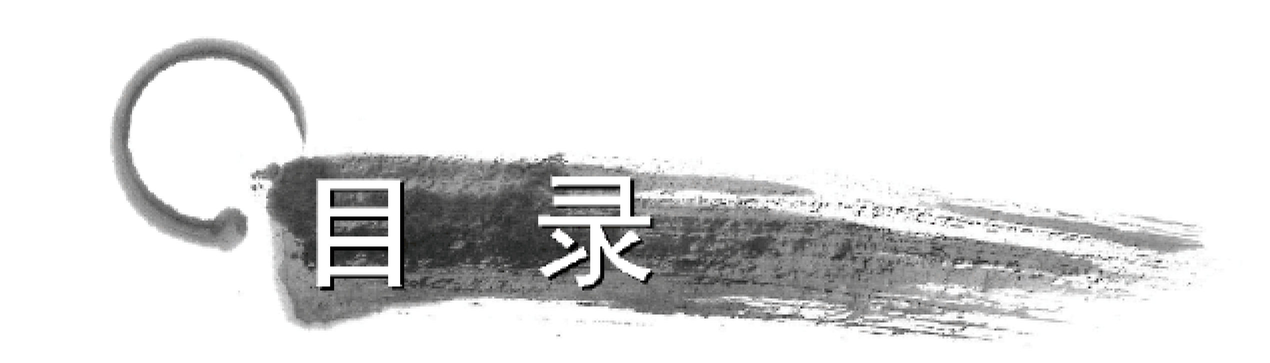
本书通俗易懂，实例丰富，既详细讲述了软件工程的基本思想和方法，又配合实例对一些关键技术问题进行了深入研究，非常适合作为高等学校计算机及相关专业本科生“软件工程”课程的教材，也适合软件项目开发人员、维护人员、管理人员自学和参考，读者可根据自己的需要对本书部分章节内容有选择地进行学习。

渤海大学的李春杰、崔红霞、刘维学、赵立双、彭霞、陈亮、李哲、李仲秋、朱萍、胡洁等教师和研究生以及中国科学院的邢涛副研究员参加了本书的编写和校对工作，在此一并表示感谢。

由于作者水平有限，加之时间仓促，书中难免存在遗漏、欠缺和错误，敬请广大读者不吝赐教。

编　者

2012 年 3 月



技 术 篇

管 理 篇

实　验　篇

技术篇

技术篇讲述与软件项目开发和维护有关的技术问题。在介绍软件工程相关概念和方法的基础上，讲述软件开发过程模型，然后按照软件生命周期过程各阶段的主要工作，分七章分别讲述可行性研究、需求分析、概要设计、详细设计、软件实现、软件测试、软件维护。

第1章 软件工程概述

1.1 软件及其特征

1.1.1 软件的定义

狭义的软件是指通过下载存储在计算机装置中的数码化比特(bit);广义的软件则包括所有根据用户指令、指示或要求的运营体系、系统或设备的支持系统,这些支持系统可以是能够实施不同任务或功能的软件包、信息或影音产品。日常生活中所讲的软件主要指计算机软件,通过对硬件设备或系统发出相应的指令,满足使用者的任务要求。

1. 根据构成软件的基本要素定义

从构成软件的基本要素来看,软件是与计算机系统操作有关的程序、规程、规则及任何与之有关的文档。软件的概念是逐渐发展起来的,在早期软件指计算机程序,此后将文档也归入其中,进一步发展包含了程序、规程、规则和文档,并强调文档是软件的重要组成部分。

(1) 程序是按具体要求产生的、适合计算机处理的指令序列。程序是软件的重要组成部分,但绝不是软件的全部。

(2) 规程是"为解决某一问题而采取的动作和经过的描述"或"每次完成某一任务时要遵循的一组工作步骤",主要描述在软件生命周期中应用和实施的有关政策、规则和标准。

(3) 规则是软件开发人员在开发软件时共同遵守的准则和法规。

(4) 文档是一种数据媒体及其记录的数据,具有永久性并可以由人或机器阅读,通常仅用于描述人工可读的内容。

2. 根据软件的组织结构定义

从软件的组织结构来看,软件是由计算机软件配置项、计算机软件部件和计算机软件单元构成的层次结构。

(1) 计算机软件配置项是为独立的配置管理而设计的,且能满足最终用户功能的一组软件。

(2) 计算机软件部件是计算机软件配置项中一个明确的部分,可以进一步分解为其他计算机软件部件和计算机软件单元。

(3) 计算机软件单元是计算机软件部件设计中确定的、能单独测试的部分。

1.1.2 软件的特征

软件具有智能性、无形性、抽象性、系统性、泛域性、依附性、非损性、复制性和演化性等特征 。

(1) 智能性。软件是人类智力劳动的产物。软件开发是高度的智能活动,软件中的程序、流程、算法、数据结构等通过人的思维进行设计、编排和组织。虽然机械、建筑、化工、纺织等所有人类生产和加工的产品都蕴藏着人的智慧,但与之相比,软件的智能性特征更为集中和明显。软件是对人类智能劳动的代替和延伸,本来需要人的智力参与的工作可由软件代替完成。软件在数据统计、科学计算、事务处理、辅助决策等方面的应用,都是对人类智能劳动的代替和延伸。

(2) 无形性。软件不像可见产品那样具有明显的物理形体和形态,人们无法直接观察到其形态。有人把计算机系统中的硬件和软件比作人的大脑和思维,大脑的生物组织相当于硬件,思维和意识则相当于软件。没有思维的大脑不具有大脑的功能,所以思维和意识是人类大脑的重要组成部分。同理,没有软件的计算机系统无法赋予计算机系统功能,所以软件是计算机系统的重要组成部分。软件的无形性基于计算机系统的光电特性。电子和绝大部分光处在人的感觉空间之外,人的视觉、听觉无法直接感觉到这些物质。软件存储在光、电、磁等介质之中,人们无法直接观察到其形态,这就增加了认识和理解的困难。

(3) 抽象性。软件属逻辑体而非物理实体,具有抽象性。首先,软件的无形性和智能性使得软件难以被认识和理解,这是软件在认识上的抽象性;其次,在软件研制过程中,需要进行调研和分析,需要对软件进行逻辑设计和组织,大量运用到抽象性思维和抽象方法,这是软件开发的抽象性;最后,软件在运行中通过输入输出界面与外部进行交流,窗口、菜单、控件等界面只是表现软件功能和作用的一种外在形式,软件丰富内涵被蕴涵在系统内部,这是软件实体的抽象性。软件的抽象性也增加了人们理解和开发软件的难度。

(4) 系统性。软件是由多种要素组成的有机整体,具有显著的系统特征。软件有确定的目标、功能和结构,软件服务的业务领域和运行的软硬件平台是软件的环境,环境约束并影响软件的功能和性能。软件的系统性还体现在需要用系统方法来看待软件及软件开发。

(5) 泛域性。泛域性是指软件可以服务于各种行业领域,只要存在人类认识可以涉足的领域和范围,软件都可以发挥作用。现在已经没有哪一个行业和领域不使用计算机系统,而使用计算机系统必然要运用软件。科学计算、事务处理、社会管理、智能决策都要使用软件,所以软件服务面向自然、社会和思维的各个领域。软件对各个领域的服务还体现在软件与所服务领域知识的结合性。软件不像其他产品,以产品本身的独立性服务于应用领域,如电话、书刊等产品从厂家的车间生产出来后直接应用于所需要的领域,而软件是对领域的智力、知识、信息处理性服务,必须把服务领域中的知识、过程、业务、方法、技术、信息等内容结合起来。所以,软件与业务领域不仅是服务关系,而且要与领域知识结合、渗透和相融。这种关系决定了软件的复杂性和软件生产的困难性。开发软件不仅要考虑软件本身的问题,而且要考虑软件服务领域中的知识、过程、业务、方法、技术、信息等问题,这些问题常常是软件开发的难点。

(6) 依附性。软件不像其他产品和设备能够独立存在和工作,软件的开发和运行必须

依附于运行环境。运行环境由计算机系统硬件、通信网络、支撑软件等要素构成。软件的依附性决定了在软件开发过程中必须考虑运行环境以及运行环境对软件的制约和影响。

(7) 非损性。软件在使用过程中不存在损耗和老化现象，只要硬件环境不发生故障和变化，软件可以永远使用。

(8) 复制性。由于软件存储在光、电、磁等介质上，所以软件可以复制，且对原软件没有任何影响。软件的可复制性特征决定了软件开发成本主要体现在软件首次开发过程中，软件一旦开发出来，复制和传播的费用一般较低。任何一批产品的生产都需要经过完整的从原材料到成品的加工过程，软件则不同。

(9) 演化性。软件投入运行后，功能、性能、界面、硬件环境等都处于不断变化之中，把软件在生命周期中不断变化的特性称为软件的演化性，也可以称为软件的易变性。软件的演化性是因为软件所处的环境不断变化、人们对软件的需求不断变化、计算机技术不断变化，因此，软件需要随着环境、需求和技术的变化而变化。软件的演化性也决定了软件在整个生命周期中要不断地改进和完善，这就是软件维护的工作。

1.2　软件危机

软件是计算机系统的灵魂，直接影响着计算机的使用和发展。20 世纪 60 年代末，随着计算机硬件技术的进步及元器件质量的逐步提高，整机的容量、运行速度及工作可靠性都有了明显提高，硬件生产成本显著下降，使计算机得到了日益广泛的应用。但是软件开发仍处于“手工作坊”阶段，软件质量主要取决于开发人员的程序设计技术，软件技术的发展不能满足人们对软件的需求，成为妨碍计算机进步的瓶颈，引发了所谓的“软件危机”。

软件危机的典型事例是 IBM 公司 1963 年至 1966 年为 IBM 360 计算机开发的 OS360 操作系统。开发期间，每年在该项目上花费约为 5000 万美元，总共投入了 5000 人年的工作量，最多时有 1000 人投入到开发工作中，共计编写了近 100 万行源程序。但由于系统过于庞大，OS360 极不可靠，每次修改后的新版本都大约存在 1000 个左右的错误。该项目负责人在总结开发过程中的沉痛教训时指出：“……正像一只逃亡的野兽在泥潭中做垂死挣扎，越是挣扎，陷得越深。最后无法逃脱灭顶之灾，……程序设计正像这样的泥潭，……一批程序员被迫在这样的泥潭中挣扎，……谁也没有料到竟会陷入这样的困境。”这个反映软件危机的典型事例成为软件技术发展过程中的一个历史性标志。

1.2.1　软件危机的表现

软件危机是在计算机软件开发和维护过程中所遇到的一系列严重问题，主要表现为以下几个方面：

(1) 对软件开发成本和进度估算不准确。由于软件的特殊性，不同类型软件开发需要的工作量、成本差异很大，常常出现实际成本比估算成本高出很多、实际进度比计划进度拖延时间长的现象，从而降低了开发商信誉，引起用户不满。

(2) 用户对已完成的软件不满意的现象经常发生。由于在开发初期，软件需求不明确，开发过程中又未能及时与用户交换意见，在需求没有得到用户确认的情况下就急于编写代

码，致使开发出的软件不能满足用户需求，甚至无法使用。

(3) 软件可维护性差。在软件开发过程中，没有使用合理、统一的方法和规范性指导，设计和实现过程的资料不完整，致使软件出现问题后难以维护。

(4) 软件质量不可靠。软件质量保证技术没能坚持不懈地应用到软件开发全过程中，测试阶段的工作不充分，提交给用户的软件质量差，在运行中暴露出大量问题。

(5) 软件可移植性差。软件可移植性是指一种计算机上的软件转移到其他计算机上的能力。从狭义上讲，可移植软件应独立于计算机的硬件环境；从广义上讲，可移植软件还应独立于计算机的软件，即高级的标准化软件，功能与机器系统结构无关，可跨越机器界限。但目前开发的大部分软件都依赖于具体的计算机系统，可移植性差。

(6) 软件产品供不应求。软件开发生产率提高的速度远远跟不上日益增长的软件需求，满足不了社会发展的需要，软件产品"供不应求"的现象使人们不能充分利用现代计算机硬件提供的巨大潜力。

(7) 软件产品价格昂贵。随着微电子学技术的不断进步和生产自动化程度的日益提高，硬件成本逐年下降。但软件开发需要大量人力，随着软件规模和数量的不断扩大，软件成本在整个计算机系统中所占的比例急剧上升，软件已成为许多计算机系统中花费最多的部分。

(8) 软件缺乏适当的文档资料。文档是软件的重要组成部分，是保证软件质量的基础性材料，但很多软件开发人员并没有写出正确的文档资料或没有进行及时更新。缺乏文档资料或文档资料不合格，必然给开发和维护带来很多困难和问题。

1.2.2 产生软件危机的原因

综合软件危机的种种表现以及软件产品作为逻辑产品的特征，将产生软件危机的主要原因归纳如下：

(1) 软件开发无计划性。造成软件项目失败的主要原因是没有计划或计划不周密。由于存在"计划没有变化快"的思想，根本不制定计划；缺乏软件开发经验和相关软件开发数据的积累，难以制定计划；对计划抱有消极态度，即使制定了计划也不被重视；管理人员缺乏对计划的监督和管理，随意对计划进行修改，必要的计划修改没有经过严格的审批程序就生效。

(2) 用户需求不明确。在软件开发过程中，用户需求不明确主要体现在三个方面：一是在软件开发出来之前，用户本身不清楚软件的具体需求；二是用户对软件开发需求的描述不精确，存在遗漏、二义性，甚至错误；三是在软件开发过程中，用户还会提出修改软件功能、界面、支撑环境等方面的要求。

(3) 需求分析不充分。需求分析是软件设计的依据。需求分析人员对用户需求的理解与用户的要求存在差异，需求分析人员不理解领域知识或不理解用户需求，需求分析人员对需求描述不清晰等，都会造成需求分析不充分，致使一些问题不能及时解决而隐藏下来，在开发后期集中暴露，造成难以挽回的损失。

(4) 缺乏正确的理论指导。即缺乏方法学和工具方面的有力支持。由于软件开发不同于大多数其他工业产品生产，其开发过程是复杂的逻辑思维过程，很大程度上依赖于开发人员高度的智力投入。过分地依靠程序设计人员在软件开发过程中的技巧和创造性，加剧了

软件开发产品的个性化,也是造成软件危机的一个重要原因。

(5) 开发过程无规范。没有对软件生命周期各阶段的工作做出合理而又统一的规定或规范,造成开发人员各行其是、不重视文档编写、设计和实现过程的文档不完善、忽视对各部分接口的界定等问题,发现错误后进行局部修补,结果是越修补错误越多,软件越来越难以维护。

(6) 软件规模越来越大。随着软件应用范围的扩大,开发规模越来越大。大型软件项目需要组织开发团队共同完成,团队成员之间的信息交流不及时、不准确,有时还会产生误解。团队管理者不能有效地、独立自主地处理大型软件开发的全部关系和各个分支,因此容易产生疏漏和错误。

(7) 软件开发复杂度越来越高。随着软件规模越来越大,复杂性也急剧增加。软件产品的特殊性和人类智力的局限性导致人们无力处理"复杂问题"。"复杂问题"的概念又是相对的,一旦采用先进的组织形式、开发方法和工具提高了软件开发的效率和能力,又会有新的、更大的、更复杂的问题摆在人们面前。

(8) 缺乏软件评测手段。由于缺乏有效的软件评测手段,软件开发人员未能在测试阶段做好充分的测试工作,造成提交给用户的软件产品质量低劣。

1.3 软件工程

1.3.1 软件工程的概念

软件危机的根本原因是软件开发和生产过程采用"手工作坊"模式,将软件开发和生产过程与程序编制混为一谈,甚至将软件与程序混为一谈。因此,解决软件危机的根本出路是软件工程,即采用如下方法:

(1) 改变过去那种手工作坊式的开发方式,采用与机械工程类似的工程方法,按照工程化的概念、原理、技术和方法来组织软件开发。

(2) 推广使用在实践中总结出来的、成功的软件开发技术和方法,研究探索更为有效的技术和方法,以提供关于软件开发的一般原则、工作框架、开发策略和实用技术。

(3) 开发并使用行之有效的软件开发工具和环境,以提高软件开发人员的工作效率,减少人为差错出现的可能性。

采用工程的概念、原理、技术和方法来开发与维护软件,把经过时间考验而证明正确的管理技术和当前能够得到的最好技术、方法与工具结合起来,这就是人们所说的"软件工程"。

软件工程是软件开发、运行、维护和引退的系统方法。软件工程包括以下三个要素:

(1) 方法与技术。提供关于软件开发的一般原则、工作框架、开发策略和实用技术。其中包括生命周期模型、自顶向下方法、结构化开发方法、面向对象开发方法、需求管理、测试技术等。

(2) 工具与环境。为软件工程方法提供自动化或半自动化的软件支撑工具和环境,对于提高软件开发生产率、保证软件质量、便于软件测试和集成、易于维护、提高软件开发过程的可见性和可控性等方面具有重要作用。

(3) 管理与标准。将软件工程的方法和工具结合起来,以保证合理、及时地进行软件开发。软件工程管理主要包括项目、配置、文档、质量、成本、人员、进度等。软件工程标准是为软件开发过程以及软件产品规定的共同准则,包括基础标准、产品标准、方法与技术标准、管理与组织标准等。

1.3.2 软件工程的基本原理

自 1968 年在联邦德国召开的国际会议上正式提出并使用"软件工程"这个术语以来,软件工程专家学者们陆续提出 100 多条有关软件工程的准则或"信条"。1983 年,著名软件工程专家 B. W. Boehm 综合了这些学者们的意见并结合 TRW 公司多年的软件开发经验,提出了七条获得公认的软件工程的基本原理:

(1) 分阶段的生命周期计划严格管理。这条原理对应软件开发的计划管理。将软件生命周期划分为若干阶段,相应地制定科学周密、切实可行的计划,并严格按照计划进行管理。在软件研制过程中,软件开发人员必须严格按照计划进行软件开发工作,各类管理人员根据计划对软件项目的研制工作进行监督和管理,严格控制对计划的随意修改,必要的计划修改必须经过严格的审批程序才能生效。

(2) 坚持进行阶段评审。这条原理对应软件开发的阶段评审。据统计,软件中的大部分缺陷是在编码之前造成的,因设计不当而引入缺陷占整个软件开发阶段引入缺陷的 50%~65%,而软件评审技术可以发现其中 75%左右的设计缺陷。因此,每个阶段都应进行严格的评审,以便及时发现并改正软件中的缺陷。评审工作要严肃认真,并由专门的评审小组进行。任何阶段工作未经过评审或者评审未获通过,均不得开展下一阶段工作。

(3) 实行严格的产品控制。这条原理对应软件开发的配置管理。在软件研制过程中,各阶段产品并非固定不变,设计规格说明、程序、需求规格说明等都可能因某些原因而更改。软件配置管理是一种标识、组织和控制更改的技术,目的是提高生产率。因此,在软件研制过程中,为了保持软件配置的一致性,必须实施软件配置管理以便进行严格的产品控制,对更改进行严格的控制和管理。

(4) 采用现代程序设计技术。这条原理对应软件开发的方法与工具。自从提出软件工程以来,人们一直致力于研究各种新的软件工程技术和方法,这些技术和方法的使用提高了软件开发和维护的效率。与此同时,人们也研制了各种支持和辅助软件开发与维护的工具和环境,这些工具和环境的使用提高了软件的开发效率、维护效率和质量。因此,在软件研制过程中,要尽量采用经过证明的、先进的开发方法和工具。

(5) 结果应能清楚地审查。这条原理对应软件开发的文档编制。软件文档是与软件研制、维护和使用有关的材料,是以人们可读的形式表示的技术数据和信息。软件文档规定了软件设计的细节,说明了软件实现的功能,描述了软件的使用方法。软件文档和计算机程序共同构成完成特定功能的软件。因此,在软件研制过程中,要重视文档编写,以确保每一阶段的结果都能够清楚地审查。

(6) 开发小组的人员应少而精。这条原理对应软件开发的人员组织。开发小组人员的素质和数量是影响软件开发效率和软件产品质量的重要因素。高素质人员比低素质人员的开发效率可能高出几倍乃至数十倍,而且高素质人员比低素质人员开发的软件中的缺陷明显减少。另外,随着小组人数的增加,可能造成开发效率和软件质量的降低。因为随着开发

小组人数的增加，通信开销和通信错误将急剧增加。

(7) 承认不断改进软件工程实践的必要性。这条原理对应软件开发过程的不断改进。遵循上述六条基本原理进行软件开发，既不能保证软件研制过程赶上时代前进的步伐，也不能保证软件研制过程跟上技术的不断进步。因此，在软件研制过程中，应承认不断改进软件工程实践的必要性，不仅要积极主动地采纳新技术，而且要不断总结经验，收集有关进度、资源耗费和软件质量等方面的数据。这些数据不仅可以评价新技术的效果，而且还可以指明应重点开发的软件工具和优先研究的软件技术，并指导软件研制过程的不断改进。

1.3.3　软件工程的目标

软件工程的目标是在给定成本、进度的前提下，开发出具有可修改性、有效性、可靠性、可理解性、可维护性、可重用性、可适应性、可移植性、可追踪性和可互操作性并且满足用户需求的软件产品。追求这些目标有助于提高软件产品的质量和开发效率，减少维护困难。

(1) 可修改性(modifiablity)：允许对系统进行修改而不增加原系统的复杂性，支持软件的调试与维护。这是一个难以达到的目标。

(2) 有效性(efficiency)：软件系统能最有效地利用计算机系统的时间资源和空间资源。在追求时间有效性和空间有效性方面经常会发生矛盾，这就需要牺牲时间效率换取空间有效性或牺牲空间效率换取时间有效性。

(3) 可靠性(reliability)：防止因概念、设计和结构等方面不完善造成的软件系统失效，具有挽回因操作不当造成软件系统失效的能力。如果可靠性得不到保证，一旦出现问题就可能是灾难性的。因此在软件设计、编码和测试过程中，必须将可靠性放在重要地位。

(4) 可理解性(understandability)：系统具有清晰的结构，能直接反映问题的需求。可理解性有助于控制软件系统的复杂性，并支持软件的维护、移植或重用。

(5) 可维护性(maintainability)：软件产品交付用户使用后，能够进行修改，以便改正潜在的错误，改进性能和其他属性，使软件产品适应环境变化。软件维护不可避免，可维护性是软件工程中一项十分重要的目标，软件的可理解性和可修改性有利于提高软件的可维护性。

(6) 可重用性(reusebility)：概念或功能相对独立的一个或一组相关模块定义为一个软件部件，软件部件可以在多种场合应用的程度称为部件的可重用性。可重用性有助于提高软件产品的质量和开发效率，降低开发和维护费用。一般来说，重用的层次越高，带来的效益越大。

(7) 可适应性(adaptability)：软件在不同的系统约束条件下，使用户需求得到满足的难易程度。适应性强的软件应采用广为流行的程序设计语言编码，在广为流行的操作系统环境中运行，采用标准术语和格式书写文档。适应性强的软件较容易推广使用。

(8) 可移植性(portability)：软件从一个计算机系统或环境转移到另一个计算机系统或环境的难易程度。为了获得比较高的可移植性，软件设计时通常采用通用的程序设计语言和运行环境支撑。可移植性支持软件的可重用性和可适应性。

(9) 可追踪性(tracebility)：根据软件需求对软件设计、程序进行正向追踪，或根据程序、软件设计对软件需求进行逆向追踪的能力。软件可追踪性依赖于软件开发各个阶段文档和程序的完整性、一致性和可理解性。降低系统的复杂性能提高软件的可追踪性。

(10) 可互操作性(interoperability):多个软件元素相互通信并协同完成任务的能力。为了实现可互操作性,软件开发通常要遵循某种标准,支持折中标准的环境将为软件元素之间的可互操作提供便利。可互操作性在分布计算环境下尤为重要。

1.4 软件工程方法学

软件工程方法学是软件开发的系统化方法,是一套完整的软件开发技术,包括原则、方法、过程和工具,是每一阶段活动、产品、验收的步骤和完成准则。软件危机的出现促进了软件工程方法学的形成和发展。大型软件系统开发需要包括分析、设计、编程、测试、维护在内的一整套的软件工程理论与技术体系,软件开发是对问题域的认识和描述。在认识事物方面,软件工程方法学在分析阶段提供了对问题域的分析和认识方法;在描述事物方面,软件工程方法学在分析和设计阶段提供了从问题域逐步过渡到编程语言的描述手段。软件工程方法学应用最广泛的是结构化方法和面向对象方法。

1.4.1 结构化方法

结构化方法(Structured Method,SM)是传统的软件开发方法,其基本思想是:用系统工程的思想和工程化的方法,按照用户至上的原则,结构化、模块化、自顶向下地对系统进行分析和设计,把一个复杂问题的求解过程分阶段进行,使得每个阶段处理的问题都控制在容易理解的范围内。

1. 结构化方法模型

结构化方法将整个系统开发过程划分为若干个相对独立的阶段,如系统规划、系统分析、系统设计、系统实施等,前三个阶段坚持自顶向下地对系统进行结构化划分。在系统规划阶段,从最顶层的管理业务入手,逐步深入到最底层;在系统分析阶段,提出新系统方案和系统设计时,从宏观整体考虑入手,先考虑系统整体优化,再考虑局部优化;在系统实施阶段,坚持自底向上逐步实施,也就是说,从最底层模块做起,按照系统设计结构,将模块一个一个地拼接到一起进行调试,自底向上、逐步地构建整个系统。结构化方法将软件开发阶段再进一步细分,其模型如图 1.1 所示。

图 1.1 结构化方法模型

2. 结构化方法过程

结构化方法分为以下三个过程。

(1) 结构化分析(Structured Analysis,SA):旨在减少分析过程中的错误,建立满足用户需求的系统逻辑模型。结构化分析使用数据流图、数据字典、结构化语言、判定表和判定树等工具,建立一种新的、称为结构化说明书的需求规格说明书。要点是面向数据流的分解和抽象,把复杂问题自顶向下逐层分解,经过一系列的分解和抽象,到最底层时很容易被描

述和实现。

(2) 结构化设计(Structured Design,SD):是一种面向数据流的设计方法,目的是确定软件结构。结构化设计是对程序结构、数据结构、过程细节和接口细节等逐步细化、评审和编写文档的过程。从技术角度上讲,软件设计分为体系结构设计、数据设计、过程设计、接口设计四个方面。从管理角度上讲,软件设计分为概要设计和详细设计两个阶段。

(3) 结构化实现(Structured Programming,SP):是进行以模块功能和处理过程设计为主的详细设计,是软件发展的一个重要里程碑。其主要观点是采用自顶向下、逐步求精的程序设计方法;使用三种基本控制结构(顺序、选择、循环)构造程序,任何程序都可由这三种基本控制结构构造。

3. 结构化方法的特点

结构化方法具有以下特点:

(1) 严格的分阶段计划。将软件开发过程从时间上分解,降低开发的复杂度。随着软件规模越来越大,开发周期越来越长,整个开发过程也越来越复杂、越来越混乱。人们解决问题的惯用方式就是分解,把复杂的问题分解成多个小部分,再逐一解决。结构化方法将完整的软件开发过程从时间上划分为许多相对独立的阶段,每一阶段都有确定的任务和方法,以及完成任务的结果和标准。

(2) 瀑布式开发模式。瀑布式开发模式按顺序实施。瀑布的特点是水流自上而下,一落到底,具有不可逆性。结构化方法采用从上到下、各个阶段逐一进行的软件开发方式,前一个阶段结束后形成的结果作为下一个阶段开始的依据。前一个阶段的任务没有按质量完成时,不进行下一个阶段的工作,每个阶段完成后不再反复。

(3) 严格的技术审查和管理复审。技术审查和管理复审是从技术和经济两个方面管理软件开发过程、保证软件质量、控制开发成本和进度。技术审查包括文档齐备性的形式审查和阶段结果的实质审查,要点是不将错误带入下一个阶段,以满足瀑布式开发模式的需要。技术审查通过后进行管理复审,管理复审由项目管理人员进行,从进度和成本的角度审查本阶段工作,并对后续阶段的成本和进度计划进行调整,对后续的开发目标进行决策。

(4) 各阶段采用结构化技术。结构化技术的基本思想是自上而下、逐步求精。结构化技术起源于编程领域,随着软件开发技术的发展被全面应用到软件开发过程的各个阶段。在本书的后续内容里将详细讲述结构化技术的具体方法和应用。

1.4.2 面向对象方法

面向对象方法(Object-Oriented Method)是一种把面向对象思想应用于软件开发过程,来指导开发活动的系统方法,简称 OO(Object-Oriented)方法,是建立在"对象"概念基础上的方法学。对象是由数据和容许的操作组成的封装体,与客观实体有直接对应关系,一个类定义了具有相似性质的一组对象,而继承性是对具有层次关系的类的属性和操作进行共享的一种方式。所谓面向对象,就是基于对象概念,以对象为中心,以类和继承为构造机制,认识、理解、刻画客观世界,设计、构建相应的软件系统。

1. 面向对象方法模型

面向对象方法是以面向对象技术为核心,利用类的继承等复用方式,通过逐步细化来建立对象模型,通过迭代和演化来完成软件开发的模式。面向对象方法的出发点就是要把现实世界中事物与事物的关系用程序表示出来,并把现实世界的组织结构在计算机上重现。面向对象方法模型如图 1.2 所示。

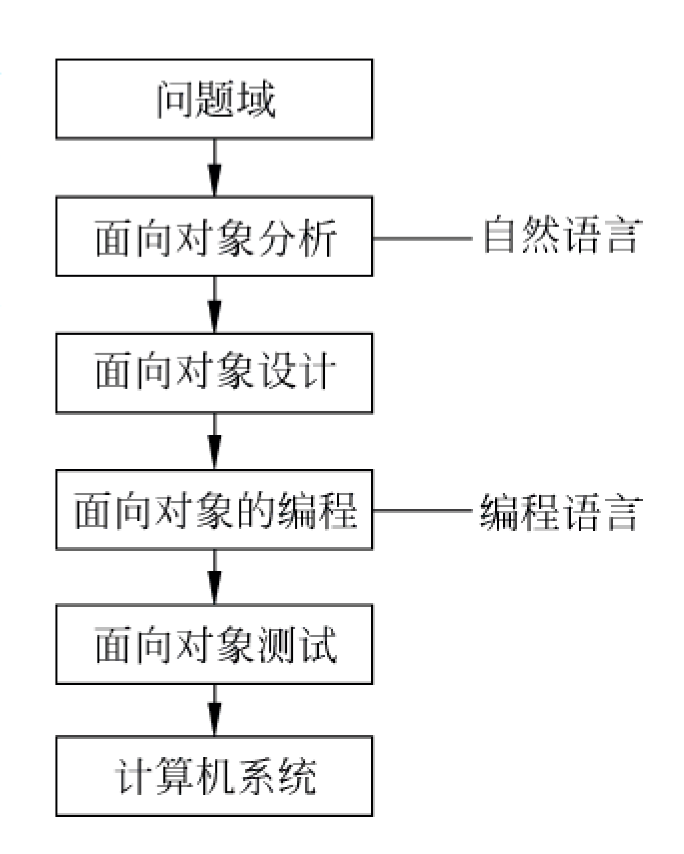


图 1.2　面向对象方法模型

2. 面向对象方法过程

在图 1.2 中,面向对象方法分为四个过程。

(1) 面向对象分析(Object-Oriented Analysis,OOA):在系统开发过程中进行了系统业务调查后,按照面向对象的思想来分析问题。OOA 与结构化分析有较大区别。OOA 强调在系统调查资料的基础上,针对 OO 方法需要的素材进行归类分析和整理,而不是针对管理业务现状和方法进行分析。

(2) 面向对象设计(Object-Oriented Design,OOD):针对系统的具体实现运用 OO 方法,OOD 与 OOA 采用相同的表示法和模型结构,这使得从 OOA 到 OOD 不存在转换,只有局部的修改或调整,并增加几个与实现有关的独立部分,因此 OOA 与 OOD 之间不存在传统方法中分析与设计之间的鸿沟,二者能够紧密衔接,降低了从 OOA 到 OOD 的难度、工作量和出错率。

(3) 面向对象的编程(Object-Oriented Programing,OOP):用一种面向对象的编程语言实现 OOD 模型中的各个成分,即用具体的数据结构定义对象属性,用具体的语句实现算法。OOP 阶段产生的程序能够紧密地对应 OOD 模型;OOD 模型中一部分对象类对应 OOA 模型,其余部分对象类对应与实现有关的因素;OOA 模型中全部类及对象都对应问题域中的事务。这样的映射关系提高了开发工作的效率和质量。

(4) 面向对象测试(Object-Oriented Test,OOT):对于运用 OO 技术开发的软件,在测试过程中继续运用 OO 技术,进行以对象概念为中心的软件测试。在测试过程中发掘并利用与 OO 方法的概念、原则及技术机制有关的语法与语义信息,可以更准确地发现程序错误并提高测试效率。对于用 OOA 和 OOD 建模并由 OOP 编程的软件,OOT 可以通过捕捉 OOA/OOD 模型信息,检查程序与模型不匹配的错误,这一点传统的软件工程方法很难达到。

3. 面向对象方法的特点

面向对象方法有以下特点:

(1) 以对象作为基本的软件构件。通过问题域的解域来模拟问题域的模型,数据和对数据的操作紧密结合。对象是一种包含了数据和操作,并且良好封装的逻辑实体。整个软件由若干个独立活动的对象构成,对象之间通过消息机制相互通信、相互作用。与面向流程的软件结构相比,面向对象的软件结构与现实世界中的实际系统更吻合,是软件构建思想上

的一次革命。

(2) 软件定义和实现的统一。无论在定义、设计还是实现阶段,面向对象方法都是通过对系统模型的逐步细化和精化来完成,因此整个软件开发过程就是一次由粗到细构建对象模型的过程,定义和实现不再截然分开。

(3) 重视软件复用。面向对象方法使用类来完成对象定义,类还可以通过继承派生出新类,因此面向对象方法本质上就具备了良好的软件复用基础,可以在不同软件中使用已有类派生出新类,既利用了前人的软件开发成果,又可以根据每个软件的实际情况产生新对象,结合对象完善的封装机制,实现对象模型的多层次利用。

(4) 通过逐步演化来完成软件开发。面向对象方法使用循环迭代方式开发软件,整个软件系统逐步精化,这既符合人类认识事物的规律,又可以尽早获得软件开发的概貌性结果,便于对软件需求进行验证。

1.4.3 敏捷方法

敏捷方法是一种针对用户需求迅速变化的现实情况,采用短周期交付形式和协作开发的轻量级软件开发方式。相对于"非敏捷",敏捷方法更强调程序员团队与业务专家之间的紧密协作、面对面的沟通、频繁交付新的软件版本、紧凑而自我组织型的团队、适应需求变化的代码编写,也更注重软件开发中人的作用。

1. 敏捷方法的特点

敏捷方法是程序员应对过多强调严格规范和文档的软件工程方法的一次更高水平上的反运动,强调以人为本、追求高目标和"轻但适用"。敏捷方法有以下特点:

(1) 追求软件交付使用这一终极目标。把一切软件开发的规范、标准、技术都置于"软件按期交付"这一目标之下,强调任何使软件能够高质量地按期交付的方法、工具和过程都应得到应用,把软件满足用户实际需求看成是软件开发的根本目标。

(2) 强调全面合作和交流。把软件开发定性为一种创造和交流的合作活动,强调用户和开发方不仅仅是合同的甲乙方,而是全面完成软件开发工作的合作方。用户和开发人员、开发人员和开发人员之间的有效交流是最重要的成功因素。敏捷方法致力于寻找快捷、方便和低成本的交流方式。

(3) 重视软件开发中人的个性。不把软件开发人员看成是完成任务的机器,强调发挥个人的潜能和主动性,保证开发人员得到足够的尊重和关怀,维持开发人员的工作积极性,以使开发工作能够有效地持续进行。

(4) 强调团队的力量。敏捷方法不认为个人能力决定软件成败,强调开发人员之间的交流和配合,强调团队的合力,以保证团队在人员调整的情况下依然能够按期完成开发工作。

(5) 灵活采用各种开发和管理方式。正因为敏捷方法强调终极目标,因此在具体开发方法和管理方式上,各种具体的方式方法虽然有许多区别,但只要能够有效地促进软件开发,各种方式方法都可以使用。

2. 敏捷方法原则

敏捷方法有以下一些原则：

(1) 最优先的目标是通过尽早交付有价值的软件来满足用户需求。

(2) 即使在开发后期也能适应需求变化，能够驾驭变化，帮助用户取得竞争优势。

(3) 频繁交付可使用的软件，时间间隔从几周到几个月，间隔越短越好。

(4) 整个开发过程中，业务人员和开发者应该在一起工作。

(5) 以积极主动的员工为主体进行软件开发，并且提供适宜环境，满足员工需要，信任员工工作。

(6) 开发团队内外最有效率和效果的信息传达方式是面对面交流。

(7) 可用的软件是进度的主要度量标准。

(8) 提倡可持续开发，管理者、开发人员和用户应始终保持稳定的节奏。

(9) 简单，尽可能减少工作量至关重要。

(10) 强调先进技术和良好设计，以不断提高敏捷性。

(11) 最好的架构、需求和设计来源于自我组织的团队。

(12) 团队定期总结提高效率的方法，并相应调整团队行为。

1.5 软件项目管理

软件项目管理是软件工程重要的研究方向。

项目管理(Project Management，PM)最早起源于美国，是第二次世界大战后期发展起来的管理技术。20 世纪 60 年代，项目管理的应用范围只局限于建筑、国防和航天等少数领域，但因为其在美国阿波罗登月项目中取得巨大成功，由此风靡全球。中国项目管理委员会对项目管理总结为："项目管理"一词具有两种含义，一是指一种管理活动，即一种有意识地按照项目的特点和规律，对项目进行组织管理的活动；二是指一种管理学科，即以项目管理活动为研究对象的一门学科，探究项目活动科学组织管理的理论与方法。

软件项目管理的对象是软件工程项目，涉及的范围覆盖了整个软件工程过程。

软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成，而对人员(people)、产品(product)、过程(process)和项目(project)进行分析和管理的活动。

软件项目管理的根本目的是为了使软件项目尤其是大型项目的整个软件生命周期(从分析、设计、编码到测试、维护全过程)都能在管理者的控制之下，以预定成本，按期、按质地完成软件，交付用户使用。研究软件项目管理，是为了从已有的成功或失败案例中总结出能够指导今后开发的通用原则及方法，避免前人的失误。

1.5.1 软件项目难于管理的原因

软件项目管理是美国在 20 世纪 70 年代中期提出的，当时美国国防部专门研究了软件开发不能按时交付、预算超支、质量达不到用户要求的原因，结果发现，70%的项目是由于管理不善引起的，而非技术原因，于是软件开发者开始逐渐重视开发过程中的各项管理。到了

20世纪90年代中期，软件研发项目管理不善的问题仍然存在。根据美国软件工程实施现状调查，软件研发的情况仍然很难预测，大约只有10％的项目能够在预定的费用和进度下交付。

1995年，美国共取消了810亿美元的商业软件项目，同时31％的项目未做完就被取消，53％的软件项目进度通常要延长50％的时间，只有9％的软件项目能够及时交付，并且费用也控制在预算之内。

软件项目难于管理，是由于软件项目管理与其他项目管理相比有很多特殊性，主要表现在以下几个方面：

（1）智力密集、可见性差。软件工程过程充满了大量高强度的脑力劳动，软件开发成果是不可见的逻辑实体，软件产品质量难以用简单的尺度度量。对于不深入掌握软件知识或缺乏软件开发实践经验的人员，不可能做好软件管理工作。软件开发任务完成得很好有时也难以体现，完成得不好有时也能制造假象，欺骗外行领导。

（2）单件生产。在特定机型上，利用特定硬件配置，由特定的系统软件或支撑软件支持，形成了特定的开发环境。再加上软件项目特定的目标，采用特定的开发方法、工具和语言，使得软件具有独一无二的特色，几乎找不到与之完全相同的软件产品。这种建立在内容、形式各异基础上的研制或生产方式，与其他领域中大规模现代化生产有很大差别，也会给管理工作造成许多实际困难。

（3）劳动密集、自动化程度低。软件项目经历的各个阶段都渗透了大量的手工劳动，这些劳动十分细致、复杂且容易出错。尽管近年来开展了软件工具和CASE工具研究，但总体来说，仍远未达到自动化程度。软件产业所处的状态，加上软件的复杂性，使得软件开发和维护难以避免出错，软件正确性难以保证，提高软件产品质量受到了很大影响。

（4）使用方法烦琐、维护困难。用户使用软件需要掌握计算机基本知识，或者接受专门培训，否则面对各种使用手册、说明和烦琐的操作步骤，学会使用要花费很大力气。另外，如果软件运行出现了问题，且没有配备专职维护人员，又得不到开发部门及时的售后服务，软件使用者将无能为力。

（5）软件工作渗透了人的因素。为高质量地完成软件项目，充分发挥人员的智力和创造精神，不仅要求软件人员具有一定的技术水平和工作经验，还要求软件人员具有良好的心理素质。软件人员的情绪和工作环境对工作有很大影响，与其他行业相比，这一特点更加突出，必须给予足够重视。

1.5.2　软件项目管理的内容与知识体系

1. 管理的内容

软件项目管理的内容主要包括人员组织与管理、软件度量、软件项目计划、风险管理、软件质量保证、软件过程能力评估、软件配置管理等。

这几个方面贯穿、交织于整个软件开发过程中，其中，人员组织与管理把注意力集中在项目组人员的构成、优化上；软件度量关注用量化的方法评测软件开发中的费用、生产率、进度和产品质量等要素是否符合期望值，包括过程度量和产品度量两个方面；软件项目计划主要包括工作量、成本、开发时间估计，并根据估计值制定和调整项目组工作；风险管理

预测未来可能出现的、危害软件产品质量的各种潜在因素，并由此采取措施进行预防；软件质量保证是保证产品和服务充分满足用户需求而进行的有计划、有组织的活动；软件过程能力评估是对软件开发能力高低进行衡量；软件配置管理是针对开发过程中的人员、工具配置、使用等提出管理策略。

2. 知识体系

软件项目管理涉及系统工程学、统计学、心理学、社会学、经济学乃至法律等方面的问题，需要用到多方面综合知识，特别是要涉及社会因素、精神因素、人的因素等，比技术问题更复杂。仅靠技术、工程或科研项目的效率、质量、成本和进度等方面很难较好地解决这些问题，必须结合工作条件、人员和社会环境等多种因素，不能简单地照搬国外的管理技术。此外，管理技术的基础是实践，为取得管理技术的成果必须反复实践。管理能够带来效率，能够赢得时间，最终将在技术前进的道路上取得领先地位。在知识爆炸、高技术迅速发展的今天，必须在战略上对待技术管理问题。

1.5.3 软件项目管理的原则

“没有规则的软件开发过程带来的只是无法预料的结果。”在软件项目管理中，有下列原则和经验可以供借鉴。

1. 计划原则

计划对软件企业非常重要，但在具体软件项目开发过程中却经常得不到重视。许多人对计划编制工作抱有消极态度，因为编制的计划常常没有用于促进实际行动。

软件项目计划是为管理工作提供合理的基础和可行的工作计划，从而保证软件项目顺利完成。为了做出具有现实性和实用性的计划，需要对项目计划过程中的工作量估算、工作结构分解、计划制定常用技术等进行分析，并遵循以下原则：

(1) 定量化原则。即确定项目任务时，尽可能定量化描述，使得每项任务的范围、时间、成本、质量、完成标准等具有明确性，可以控制和度量。

(2) 个人化原则。每个具体任务应当落实到项目组的每个成员，使得每个人都明确自己的工作和职责。

(3) 简单化原则。任务和目标的描述应当简单而直接，使得每个参与人都能明确且无二义性。

(4) 现实性原则。确定的每个任务或目标都可以实现，而不追求理想化结果。

2. Brooks 原则

向一个已经滞后的项目添加人员，可能会使项目更加滞后。因为向开发团队中加入新成员，要进行相关培训、熟悉环境等工作，人员之间的沟通路径增加，迫使项目的工作效率急剧下降。工作效率下降需要加班来进行弥补，加班造成的疲劳导致工作效率再次降低，但工作成本却不断地向上攀升。很多项目管理者并没有注意到这一点，认为“人多力量大”，当项目工作完不成时，就增加人员，最终造成恶性循环。

3. 80-20 原则

80-20 原则在软件开发和项目管理方面有许多“实例”。

实例之一，20％的工作耗费了 80％的时间。仔细分析，这些工作分为必需的和非必需的，应当压缩非必需部分，或是暂时将其搁置，不必太重视。软件项目管理的事实表明，开发人员在非必需的工作上耗费了太多精力，实际上用户并不看重这些，应在用户所关心的工作上下工夫。

实例之二，20％的人员承担了 80％的项目工作。考虑到开发人员能力的多样性，不应采取任务均分的愚蠢做法，因为就系统论的观点来看，互补结构比对等结构更稳定。此外，作为项目管理人员，了解员工的能力特点，将其放在合适的位置上，更有利于项目工作的进行。很多管理人员常常抱怨下属能力问题，究其实质，往往是管理人员未能发现下属潜能。不能以“经验”这样的思维定式做决定，而要发现人的长处，运用人的长处。

4. 默认无效原则

项目成员理解并赞成项目的范围、目标和策略吗？不少项目管理者认为“沉默意味着同意”。项目开发人员沉默并非完全赞成管理人员的意见，但实际上人们或多或少都会陷入这样的思维误区。项目管理者切不可认为沉默就是同意，沉默在很大程度上说明项目开发人员尚未弄清楚项目的范围、任务和目标。为此，项目管理者还需要同开发人员进行充分的沟通，了解开发人员的想法。在对项目没有共同一致的理解前提下，团队不可能成功。

5. 帕金森原则

英国著名历史学家诺斯古德·帕金森在他的《帕金森定律》里，阐述了组织机构臃肿、人员膨胀的原因及后果，后来，“帕金森定律”成为反映政府部门机构臃肿、人浮于事、效率低下的代名词，这在软件项目管理中同样适用。没有时间限制，工作可能无限延期。在软件开发中，如果没有严格的时间限制，开发人员往往比较懈怠。这是由人的天性决定的，千万不要指望会发生奇迹——“所有员工的思想觉悟异常崇高”。作为项目管理者，应充分考虑到员工的工作效率和计划变更带来的负面影响，制定合理的项目工期并鼓励相关人员尽快完成。机构臃肿也可能是因为管理幅度过宽，一个人的管理幅宽有限，如果一个项目需要过多人员参与，就要分成若干个组，项目经理管理若干个组长，重视管理团队建设，进行适当分权与分责，以此来提高效率。

6. 时间分配原则

在项目管理计划编制过程中，有些项目经理将资源可用率(人、设备)等设置为 100％。但是开发人员需要休息、吃饭、开会等，根本不可能把所有时间放在开发工作上，而且还要考虑到工作效率是否保持在一个恒定水平上。由于计划不合理，通常开发人员被迫拼命加班。

在实际工作中，开发人员的时间利用率能够达到 80％就已经非常高了，通常是 50％～60％左右。如果开发工作组织合理，完成开发工作需要的时间通常为原计划时间的 1.2～1.5 倍。如果开发工作组织不合理，又是新软件项目，所需要的时间通常为原计划的 2～3 倍。由于项目工期的紧迫性，很多开发人员都是加班加点，如果实在完不成，只有降低标准。开

发人员可能同时承担几个项目，也可能正在做某项工作时突然要求出差从事其他项目，以致使开发人员忙于应付、身心疲惫。管理人员在制定计划、分配工作时，应考虑到这些因素。

7. 验收标准原则

完成某项工作时，常常会为以何种结果为宜而感到困惑。不求质量则往往凭经验草草了事，追求完美则要耗费太多的精力，但此番耗费未必针对该项任务，这是由于没有完成标准而导致的结果。软件项目开发常常以验收标准为原则，只有达到验收标准，软件项目才能成功交付。作为项目经理，只有制定好每个任务的验收标准，才能够严格把好质量关，同时了解项目工作的进度情况。

8. 变化原则

项目管理中唯一不变的是什么？答案是，唯一不变的就是“变化”。软件技术发展迅速，只有变化、创新，才能开发出适合市场需求的软件，软件开发企业才有活力、有发展。但变化可能带来风险，项目管理人员不能像经济学里“风险规避原则”描述的那样怀有逃避态度，而是应该及早预测可能出现的风险，做好风险准备。

9. 软件工程标准原则

随着人们对计算机软件认识的逐渐深入，软件工作的范围也从简单的程序设计扩展到整个生命周期。软件计划制定、需求分析、设计、程序编写、测试、维护及其相应的组织管理工作都需要按一定的规范进行，因而提出了软件工程标准化的原则。

软件开发项目需要多个层次、不同分工的人员相配合，在项目的各个部分以及各开发阶段之间也都存在着许多联系和衔接问题。如何把这些错综复杂的关系协调好，需要有一系列统一的约束和规定；在软件开发项目取得阶段成果或最后完成时，需要进行阶段评审和验收测试；对于投入运行的软件，维护工作中遇到的问题又与开发工作有着密切的关系；管理工作渗透到软件生命周期的各个环节。所有这些都要求提供统一的行动规范和衡量准则，使得各项工作都有章可循。

软件工程标准化会给软件工作带来许多好处，比如提高软件的可靠性、可维护性和可移植性，提高软件生产率，提高软件人员的技术水平，提高软件人员之间的通信效率等工作，有利于减少差错和误解，有利于软件管理，有利于降低软件产品的开发成本和运行维护成本，有利于缩短软件开发周期。

10. 复用和组织变革原则——解决项目问题的未来之路

日益突出的工期、成本、质量等问题，是大多数项目管理者最为关心的问题。从实践来看，加强复用力度、建立复用体系和实施组织变革是有效途径。复用能够提高项目的生产率，降低项目风险。通过复用，项目管理者能够快速地进入项目问题定义，减少项目开发人员的工作量，从而尽可能地解决时间、资源等方面的过载问题。另外一条途径是实施项目团队的组织变革，精简管理机构，重新定义工作职责，制定柔性的工作流程，改善开发人员的沟通状况，提高人员的开发效率，努力营造良好的开发环境。这样才能从根本上解决项目开发的种种棘手问题。

结论：作为软件项目管理者，仅仅了解和运用原则是不够的，若要深入掌握项目管理的知识和技巧，还必须深入学习项目管理、管理心理学、质量管理学、组织变革、系统论等方面的知识，并在工作中不断地总结和实践。

思考题

1. 如何理解软件的定义？
2. 软件有哪些特征？
3. 软件危机有怎样的表现？
4. 产生软件危机的原因是什么？
5. 如何理解软件工程的概念？
6. 如何理解软件工程的基本原理？
7. 软件工程的目标是什么？
8. 什么是结构化方法？结构化方法包括哪些过程？结构化方法有何特点？
9. 什么是面向对象方法？面向对象方法包括哪些过程？面向对象方法有何特点？
10. 什么是敏捷方法？敏捷方法有何特点？如何理解敏捷方法的原则？
11. 软件项目难于管理的原因是什么？
12. 软件项目管理的内容主要包括哪些方面？
13. 软件项目管理的原则有哪些？
14. 如何理解软件项目管理的 80-20 原则？

第2章 软件开发过程模型

2.1 软件生命周期

软件生命周期是从设计软件产品开始到软件产品不能再使用为止的时间。典型的软件生命周期包括需求阶段、设计阶段、实现阶段、测试阶段、安装和验收阶段、运行和维护阶段，有时还包括引退阶段。软件生命周期可以划分成若干个相互独立而又相互联系的阶段，每一阶段工作以上一阶段工作的结果为依据，并为下一阶段工作提供基础。

软件生命周期的提出是为了更好地管理软件开发的步骤和方法，以及软件的维护和升级。软件生存时间可以被看做一个整体，以时间的推移和软件开发的工作重心作为划分点，把软件开发和维护工作细分为若干个相对独立的部分，从而更好地控制软件开发的进度和难度，同时也有利于降低软件的出错频率，协调各个部门间的工作配合和责任分配。

软件生命周期的各个阶段划分并没有一成不变的法则，不同的开发方式、软件种类、软件规模、开发环境，都会在不同程度上影响软件生命周期各阶段的划分。生命周期根据实际情况划分，旨在更好地利用资源（主要是人力资源、软件资源、技术资源和源码资源），降低软件开发风险、复杂度和开发成本（主要以开发时间和投入资源为衡量标准）。要更好地对软件生命周期各个阶段进行划分，必须遵循的一条基本原则就是各个阶段的任务尽可能相对独立，同一阶段各项任务的性质尽可能相同，从而降低各个阶段任务的复杂度，减少不同阶段任务之间的联系。这样对软件项目开发的组织管理十分必要，对最终软件项目开发成功也是不可缺少的。

尽管软件生命周期各个阶段的划分没有明确规则，但就一般性而言，软件生命周期包括可行性研究、开发计划、需求分析、概要设计、详细设计、代码编写、软件测试和软件维护等活动（有时把概要设计和详细设计合在一起，统称为软件设计或设计），这些活动都是软件开发过程中必须要经历的，要合理地安排到各个阶段中去。

2.2 软件过程

1. 软件过程的定义

软件过程是指软件生命周期中的一系列相关过程，是将用户需求转化为可执行系统的演化过程所进行的软件工程的全部活动，是用于生产软件产品的工具、方法和实践的集合。

软件过程中的"过程"是创建一个产品或完成某些任务的一种系统化的方法和工作过程,执行者不再仅仅是计算机,而经常是由具体承担任务的软件开发人员使用给定的开发工具来执行,甚至可以是一个无法在计算机上运行的过程,完全由人工或人工借助计算机以外的工具来完成。

软件过程是关系复杂的软件活动的集合,各活动之间有着严格密切的关系,有的是异步并行,有的是互为条件,因此实际软件过程中的软件活动存在复杂的网状关系。如何有效地对软件活动进行管理成为软件过程管理的重要内容。

软件过程是改进软件质量和组织性能的主要因素之一。M. Dowson 曾指出:"软件产品质量在很大程度上依赖于软件过程,尤其是大规模的软件开发更是如此。"因此,不少软件开发企业力图通过改进开发过程来改善软件产品质量、提高软件生产率、缩短产品的开发时间,从而增加企业的竞争力和效益。

2. 软件过程管理的必要性

在软件开发机构中,实施软件过程管理的必要性如下:

(1) 提高软件企业的开发效率和产品质量。

(2) 有效地对软件开发项目进行管理,便于按照进程和预算完成软件项目计划,实现预期的经济效益和社会效益。

(3) 有助于理解软件开发的基本原则,帮助软件机构做出正确决策。

(4) 有利于标准化开发人员的工作,提高软件的可重用性和组间协作。

(5) 改善软件机构对软件的维护。

(6) 软件过程管理机制本身是不断提高的,可以不断采用新的、更好的软件开发经验。

3. 软件过程管理的组成

根据 ISO/IEC 15504 软件过程评估标准,软件过程被分为五个过程:工程过程、支持过程、管理过程、组织过程和客户-供应商过程。基础是组织过程,核心是工程过程,关键是管理过程。组成结构如图 2.1 所示。

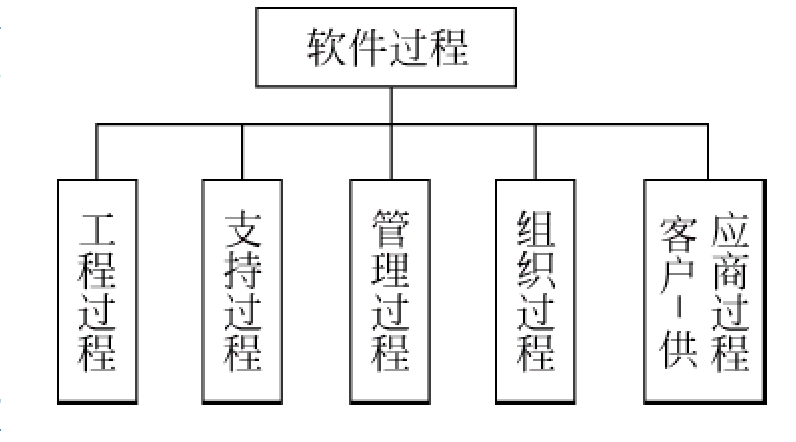


图 2.1　软件过程组成结构

(1) 工程过程(Engineering Process,ENG):软件系统或产品的定义、设计、实现以及维护的过程。

(2) 支持过程(Support Process,SUP):在整个软件生命周期中可能随时被任何其他过程所采用的、起辅助作用的过程。

(3) 管理过程(Management Process,MAN):在整个生命周期中为工程过程、支持过程和客户-供应商过程的实践活动提供指导、跟踪和监控的过程。

(4) 组织过程(Organization Process,ORG):用于建立组织商业目标和定义组织内部培训、开发活动和资源使用等规则的过程,并有助于组织在实施项目时更好、更快地实现预定的开发任务和商业目标。

(5) 客户-供应商过程(Customer-Supplier Process,CUS):直接影响到客户、对开发的支持、向客户交付软件以及软件正确操作与使用的过程。

2.3 软件开发过程

软件开发过程是以生命周期各阶段的活动划分为基础，将用户需求转化为软件系统活动集合的过程，如图 2.2 所示。

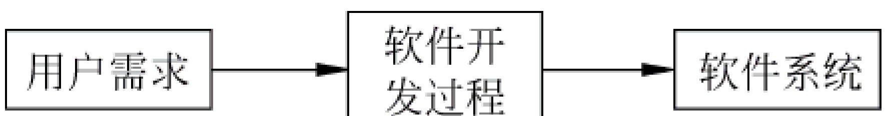


图 2.2 软件开发过程

软件开发过程包括需求分析、设计、编码、集成、测试、安装和验收等活动。收集各方面的用户需求信息（过程的输入），定义用户产品的功能和性能，通过设计将用户需求转换为软件表示，在逻辑上定义产品功能，设计结果作为编码的框架和依据，最后通过编程将设计转换成计算机可读的形式。整个软件开发过程可以进一步分为五个子过程：可行性研究、需求分析、设计、编码、测试。

软件开发过程的划分对软件管理和资源投入以及软件项目最终开发能否成功具有重要意义。要充分了解各个活动的定义和任务，以便合理、准确、客观地安排每一阶段的工作。各个阶段完成的主要工作如下。

1. 开发计划和可行性研究阶段

这两个活动通常被整合在一起，实际工作中归结到同一个阶段进行，甚至可以看成是一个整体，要回答“做什么？如何做？可不可能完成？”等问题。可行性研究要依靠开发计划提供依据，而开发计划只有在初步得到可行性后才能深入制定，两个活动是互相制约、互相促进的关系。

这个阶段要和各方多沟通，尽可能得到准确的问题定义，并确保各方的理解相同。对问题的精确理解和定义在开发计划阶段解决，更符合各方的利益要求，比在需求分析阶段解决更合理，同时不会对软件开发方向造成隐患，也不会给双方就软件开发费用达成造成不必要的麻烦。

用户提出软件开发要求后，系统分析员要对用户机构进行了解，明确是什么样的机构、主要业务是什么，对最终的软件使用部门进行观察研究、组织开会讨论等，通过这一系列工作确定软件项目的性质、目标和规模，这些工作如同需求分析的简化版，为项目的后期工作奠定基础，在此基础上写出可行性研究报告。

如果可行性研究的结论是可行的，接下来就要制定详细的开发计划。开发计划主要根据开发项目的目标、性能、功能、规模来确定需要的资源，主要包括三个方面，即硬件资源、软件资源和人力资源，还要对项目的开发费用、开发进度做出估计，供决策者和用户参考。

至此，本阶段的工作任务基本完成，将《可行性研究报告》和《项目开发计划》提交管理部门审查。

2. 需求分析阶段

需求分析阶段确定软件功能和性能要求，根据功能要求进行数据流程分析，提出系统逻辑模型，并与文字说明、图表、流程、规范等共同组成系统需求规格说明书。这一阶段的主要工作可概括为四个方面：需求获取、需求分析、编写需求规格说明书和需求评审。

(1) 需求获取：确定目标系统各方面的需求，建立获取用户需求的方法框架，并支持和监控需求获取的过程。

(2) 需求分析：对获取的需求进行分析与综合，给出系统的解决方案和目标系统的逻辑模型。

(3) 编写需求规格说明书：作为需求分析的阶段成果，为用户、分析人员和设计人员之间的交流提供方便，直接支持目标软件系统的确认，又可以作为控制软件开发进程的依据。

(4) 需求评审：对需求分析阶段的工作进行评审，验证需求文档的一致性、可行性、完整性和有效性。

3. 软件设计阶段

软件设计阶段包括概要设计和详细设计两个阶段。

(1) 概要设计阶段。在软件开发过程中，概要设计阶段通常安排在需求分析之后进行，是建立系统整体结构、进行模块划分、根据要求确定接口的过程。概要设计的主要工作是设计模块和组织模块，把需求分析中的软件功能用模块结构的形式描述出来，每个模块都有明确的意义和功能。数据库设计也是概要设计的工作之一，即软件系统要存储什么数据，以及数据的结构和关系等。

(2) 详细设计阶段。详细设计阶段把在概要设计阶段划分出来的模块要实现的功能，用相应的设计工具详细地描述出实现步骤，也就是写出算法。详细设计阶段使用的语言或图表都应该有精确和唯一的描述，不允许出现“二义性”或“多义性”。详细设计的任务是为每个模块完成的功能进行具体而精确的描述，根据功能描述再转化成精确的、结构化的软件过程描述，软件过程描述可直接对应到相应的代码，也就是程序员在下一阶段根据过程描述编写程序代码。

4. 编写代码阶段

编写代码阶段在计算机上用计算机语言实现所设计的软件功能，把过程描述翻译成程序并测试程序的正确性。编写代码时要高度对应详细设计阶段描述的算法，因为以后的维护或升级都是以详细设计的文档资料为根据。如果代码和详细设计的描述有偏差，很容易误导以后的维护工作，而且这种错误很难被发现，浪费不必要的人力物力。编程时还要注意，尽可能在重点和难点处留下注释，这样对以后的维护和修改有帮助。

5. 软件测试阶段

通过单元测试，检验模块内部的结构和功能；通过集成测试，把模块连接成系统，重点测试模块间的接口；通过确认测试，对需求分析的软件功能和性能进行测试，确认是否达到要求；通过系统测试，测试软件系统在真实系统环境中的运行状况。单元测试和集成测试是由开发者完成的；确认测试和系统测试是在用户参与下，由开发者和用户共同完成的。软件测试的方法一般分为两种：静态测试法和动态测试法，动态测试法又根据测试用例的不同而分为白盒测试和黑盒测试两种。

2.4 软件开发过程模型

软件开发作为一门独立的学科，有其自身的理论体系——软件工程。软件工程理论涉及的内容很多，其中对软件开发项目影响最大的是发展了一系列的开发过程模型，包括瀑布模型、原型模型、螺旋模型和软件包模型等。下面简要介绍几个在软件开发企业常用的过程模型。

2.4.1 瀑布模型

瀑布模型是美国人 Winston Royce 于 1970 年向 IEEE WESCON 提交的一篇名为《管理大规模软件系统的开发》的论文中首次提出的。这篇文章以他在管理大型软件项目开发时学到的经验为基础，抽象出了具有深刻见解而又简洁的软件项目开发管理方法。由于这种方法是从一个阶段成瀑布流入下一个阶段，所以称为“瀑布模型”。

瀑布模型是从时间角度对软件开发和维护的复杂问题进行分解。按软件生命周期依次划分为六个阶段：可行性研究、需求分析、软件设计、软件编码、软件测试、运行与维护。

1. 理论的瀑布模型

理论的瀑布模型结构如图 2.3(a)所示，有两重含义：

(1) 必须等前一阶段的工作完成之后，才能开始后一阶段的工作。

(2) 前一阶段的输出文档是后一阶段的输入文档，只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。

缺乏软件工程实践经验的软件开发人员接到软件项目开发任务后，常常急于求成，总想尽早开始编写代码。实践表明，对于规模较大的软件项目，往往编码开始得越早，最终完成开发工作所需要的时间反而越长。这是因为前面阶段工作错误太多，过早地进行软件编码往往导致大量返工，有时甚至造成软件工程过程失败。所以，尽可能地推迟软件编码是按照瀑布模型开发软件的一条重要的指导原则。

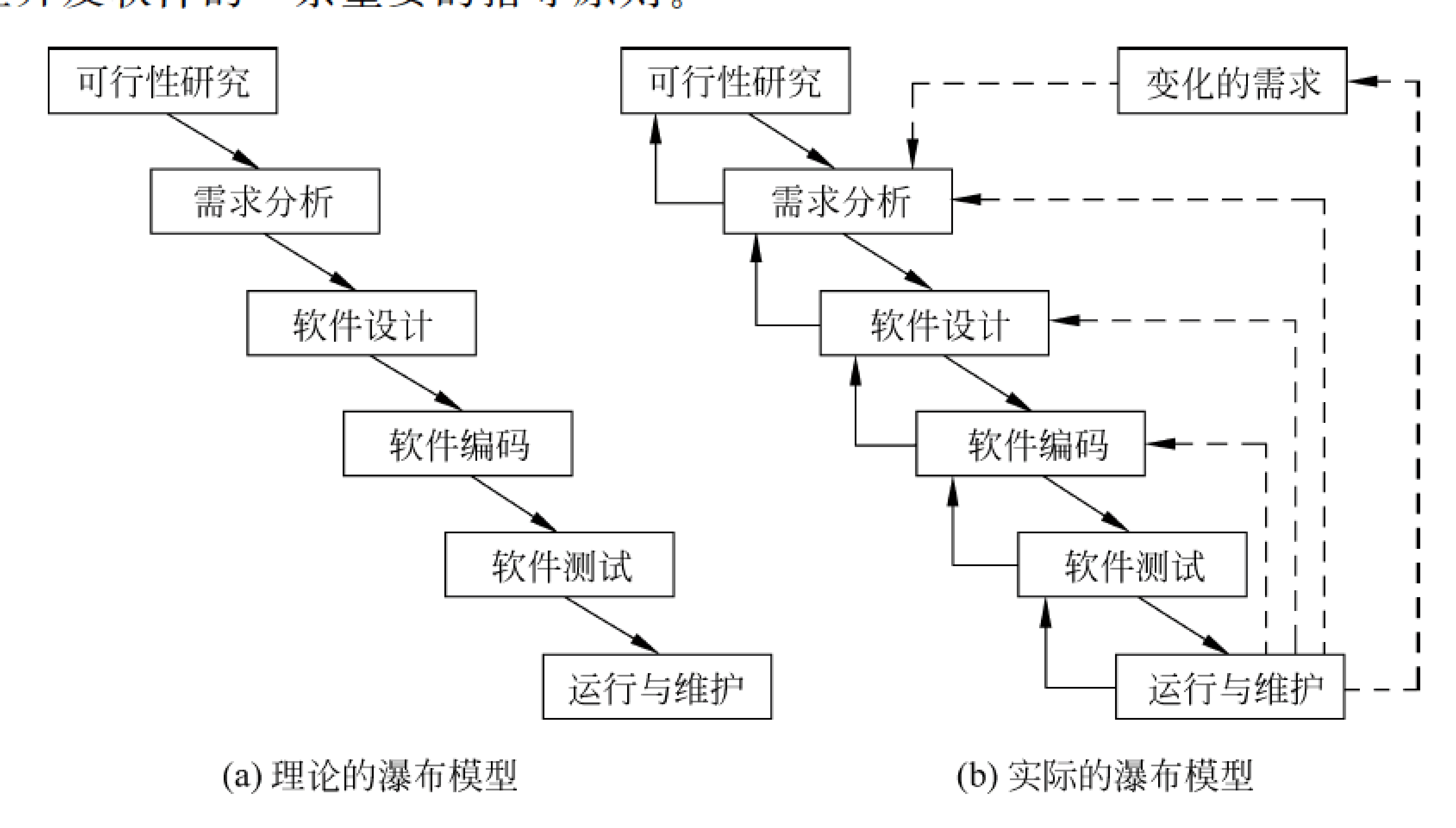


图 2.3 瀑布模型

2. 实际的瀑布模型

理论的瀑布模型过于理想化。实际上，人们在工作中不可避免地发生错误。在软件设计阶段可能发现需求说明书中的错误，而软件设计阶段的缺陷或错误可能在软件编码阶段显现出来，在软件测试阶段可能发现需求分析、软件设计、软件编码阶段的错误。因此，实际的瀑布模型带有“反馈环”，如图 2.3(b)所示。图中实线箭头表示开发过程，虚线箭头表示维护过程。当在后面阶段发现前面阶段的错误时，需要沿图中左侧的反馈线返回前面阶段，修正前面阶段的工作成果后再回来继续完成后面阶段的工作。

3. 瀑布模型总结

为了保证软件开发质量，运用瀑布模型应坚持做到以下两点：

(1) 每个阶段都完成规定的文档，没有交出合格的文档就没有完成阶段性工作。完整、准确、合格的文档不仅是开发阶段各类人员之间相互通信的媒介，也是运行阶段对软件进行维护的重要依据。

(2) 每个阶段结束前都要对提交的文档进行评审，以便尽早发现问题，改正错误。软件开发中的错误具有放大效应，越早阶段犯下的错误，发现的时间越晚，改正错误需要付出的代价也就越高。因此，及时检查是保证软件质量、降低软件成本的重要措施。

瀑布模型的优缺点和适用情况如表 2.1 所示。

表 2.1　瀑布模型的优缺点和适用情况

优点	定义清楚，应用广泛； 强迫开发人员采用规范化的方法(如结构化方法)； 严格规定每个阶段提交的文档； 易于建模和理解； 便于计划和管理； 有支持生命周期模型的多种工具
缺点	必须在开始时就知道大多数需求； 不便于适应需求的变化； 在项目接近完成之前，产品不能投入使用； 可运行的软件交付给用户之前，用户只能通过文档来了解产品
适用情况	待开发项目与以前的成功项目类似； 待开发项目的需求稳定且很好理解； 使用的技术经过验证并且成熟； 整个项目的开发周期较长(至少一年)； 用户不需要任何阶段性产品

2.4.2　V 模型

V 模型是瀑布模型的一种变体，由于整个开发过程构造成一个 V 字形而得名。V 模型强调软件开发的协作和速度，将软件实现和验证有机地结合起来，在保证较高软件质量的前提下缩短开发周期。V 模型结构如图 2.4 所示。

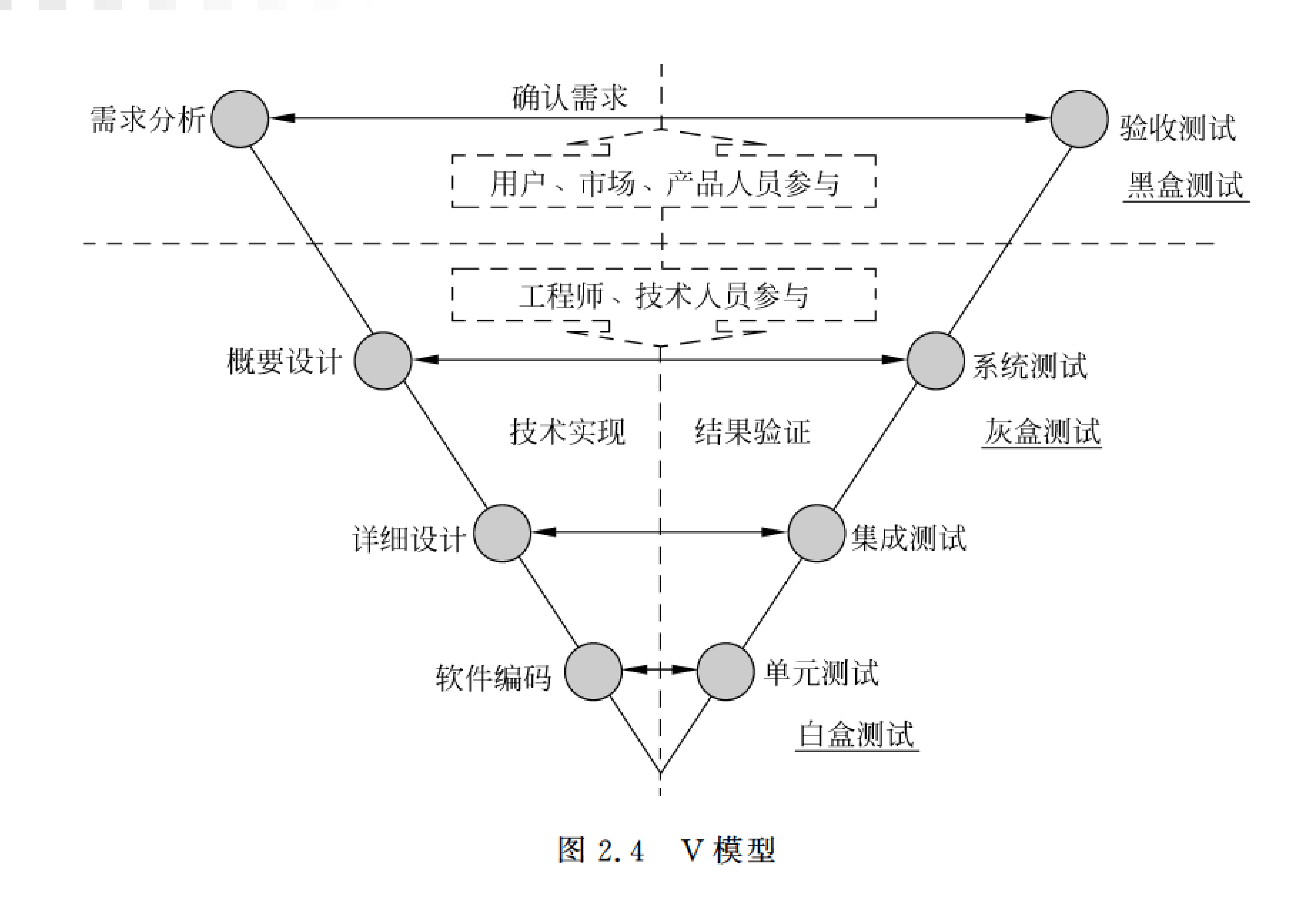


图 2.4　V 模型

下面通过对 V 模型的水平和垂直的关联和比较分析，理解软件开发和测试的关系，理解 V 模型具有面向用户、效率高、质量预防意识好等特点，理解它是可操作性强的软件开发过程模型。

1. 从水平方向看

垂直虚线左边是分析和设计，是软件设计实现的过程，同时伴随着质量保证活动——审核的过程，也就是静态的测试过程；垂直虚线右边是对左边结果的验证，是动态测试的过程，即对分析和设计的结果进行测试，以确认是否满足用户需求。左右两边的对应关系如下：

（1）需求分析对应验收测试。说明在做需求分析时，测试人员就可以阅读、审查需求分析的结果，从而了解产品的设计特性、用户的真正需求，确定测试目标，可以准备测试用例并策划测试活动。

（2）概要设计对应系统测试。说明当设计人员做概要设计时，测试人员可以了解系统是如何实现的、基于什么平台，这样可以设计系统的测试方案和测试计划，并事先准备系统的测试环境，包括硬件和第三方软件的采购。这些准备工作实际要花费很多时间。

（3）详细设计对应集成测试。说明当设计人员做详细设计时，测试人员可以参与设计，对设计进行评审，找出设计缺陷，同时设计功能、新特性等各方面的测试用例，完善测试计划，并基于测试用例开发测试脚本。

（4）软件编码对应单元测试。说明在编程的同时进行单元测试是一种很有效的方法，可以尽快找出程序中的错误，充分的单元测试可以大幅度提高程序质量、降低成本。

从水平对应关系可以看出，V 模型能使质量保证活动和项目同时展开。项目一旦启动，软件测试工作也就启动了，避免了瀑布模型所带来的误区——软件测试只能在编码完成之后进行。

2. 从垂直方向看

水平虚线上部表明，需求分析、系统定义和验收测试等工作主要是面向用户，要和用户进行充分的沟通和交流，或者和用户一起完成。水平虚线下部的大部分工作，相对来说，都是技术工作，在开发组织内部进行，主要是由工程师、技术人员完成。

从垂直方向看，越在下面，白盒测试方法使用越多；到了集成测试、系统测试，更多是将白盒测试方法与黑盒测试方法结合起来使用，形成灰盒测试方法；而在验收测试过程中，用户一般要参与，所以使用黑盒测试方法。

2.4.3　原型模型

原型模型是 20 世纪 80 年代初期，为解决瀑布模型需求理解困难、开发周期长、见效慢等问题，借助第四代程序开发语言而产生的一种软件开发方法。由于大部分系统在开发初期或者需求分析阶段，用户和软件工程师的背景知识和文化上的差异，对软件需求的理解非常困难。为此，软件开发人员先根据用户提出的软件定义，快速开发一个原型，向用户展示，然后用户根据这个原型提出修改意见，再进一步修改、完善，确认软件系统的需求并达到一致的理解。原型模型结构如图 2.5 所示。

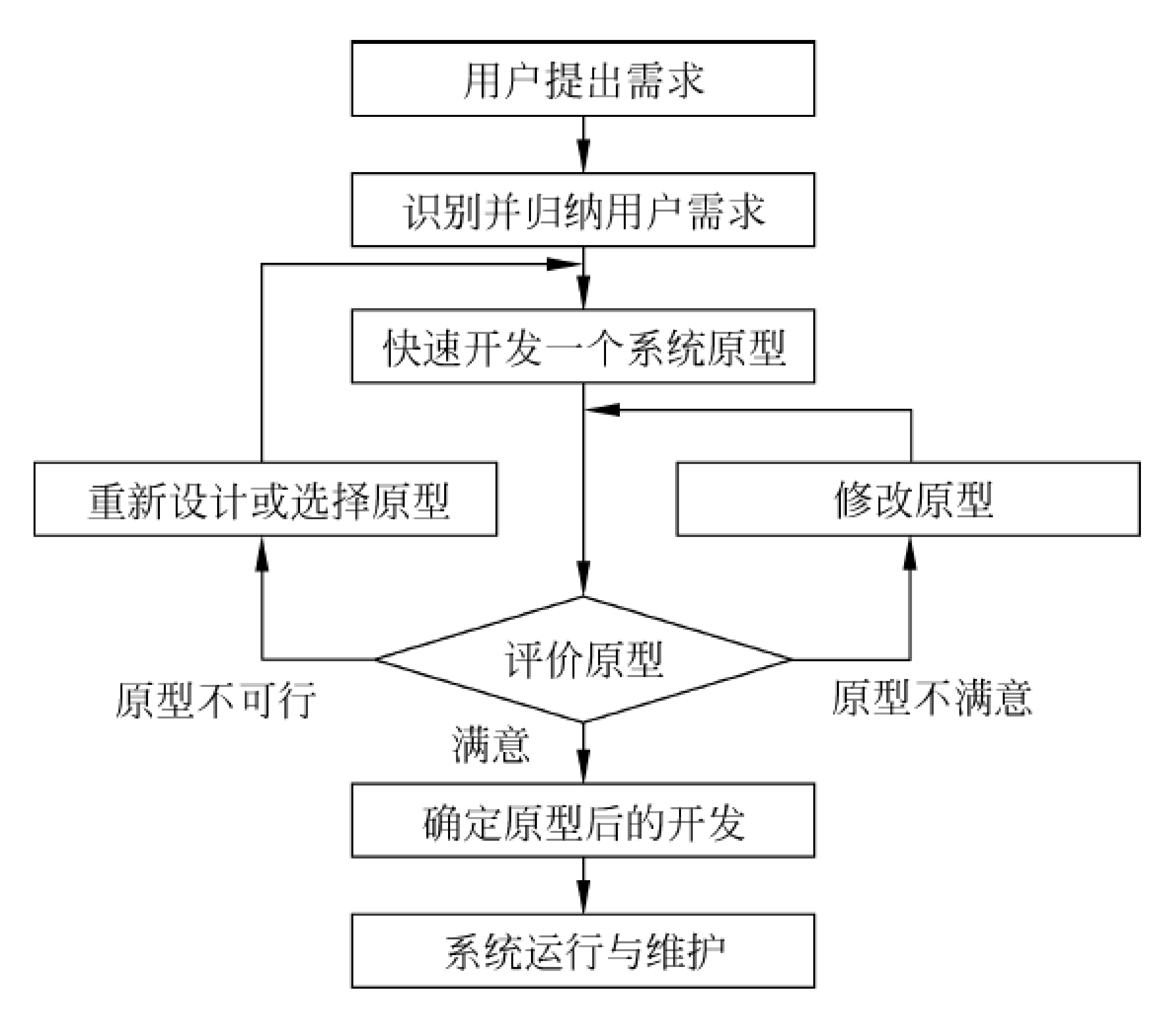


图 2.5　原型模型

1. 原型模型的基本过程

使用原型模型的基本过程是：

（1）用户和开发人员根据初始需求共同制定项目规划。

（2）用户和开发人员利用快速分析技术共同定义需求和规格。

（3）设计者开发系统原型。

（4）设计者演示系统原型，用户来评价性能并识别问题。

（5）如果原型不可行，重新设计或选择原型。如果原型不满意，修改原型。循环这个过程，直到用户满意为止。

(6) 在原型的基础上进行更详细的设计、开发和完善。
(7) 运行系统并进入系统维护阶段。

2. 原型模型的软件支撑环境

原型模型需要以下软件支撑环境：
(1) 方便灵活的关系数据库系统。
(2) 完整的程序生成软件。
(3) 与数据库对应的、方便灵活的数据字典。
(4) 可以快速抽象或者容易提炼的原型。

3. 原型模型的优缺点和适用情况

原型模型的优缺点和适用情况如表 2.2 所示。

表 2.2 原型模型的优缺点和适用情况

优点	直观形象，符合人们认识事物循序渐进的规律，容易被接受； 有效地避免开发人员和用户对需求理解的不一致性； 及时暴露问题、及时反馈，确保系统的正确性； 开发周期短、成本低，软件尽早投入使用
缺点	为了加快开发速度，常常导致软件质量的降低； 没有严格的开发文档，维护困难； 缺乏统一的规划和开发标准； 难以对系统的开发过程进行控制
适用情况	用户需求不确定或经常发生变化； 开发人员的经验不丰富； 开发规模不大、不太复杂的系统。因为大型系统不经过整体的分析和设计是不行的

2.4.4 螺旋模型

瀑布模型对每个阶段有严格的界定，因而要求在软件开发开始阶段就完全确定需求，实际在很多情况下无法实现。对于复杂的大型软件，运用原型模型，开发一个原型往往达不到要求。因而勃姆(Boehm，B. W)将瀑布模型与快速原型模型结合起来提出了螺旋模型。该模型要求不断迭代，同时要像螺旋一样不断前进，即每次迭代都不是在原水平上进行，是对整个开发过程进行迭代，而不仅仅对编码、测试进行迭代。螺旋模型结构如图 2.6 所示。

1. 工作步骤和内容

每一螺旋周期由四个步骤组成：
(1) 确定下一阶段目标、开发方案及约束条件。
(2) 风险分析、构造原型。
(3) 开发、验证阶段软件产品。
(4) 制定下一阶段计划。

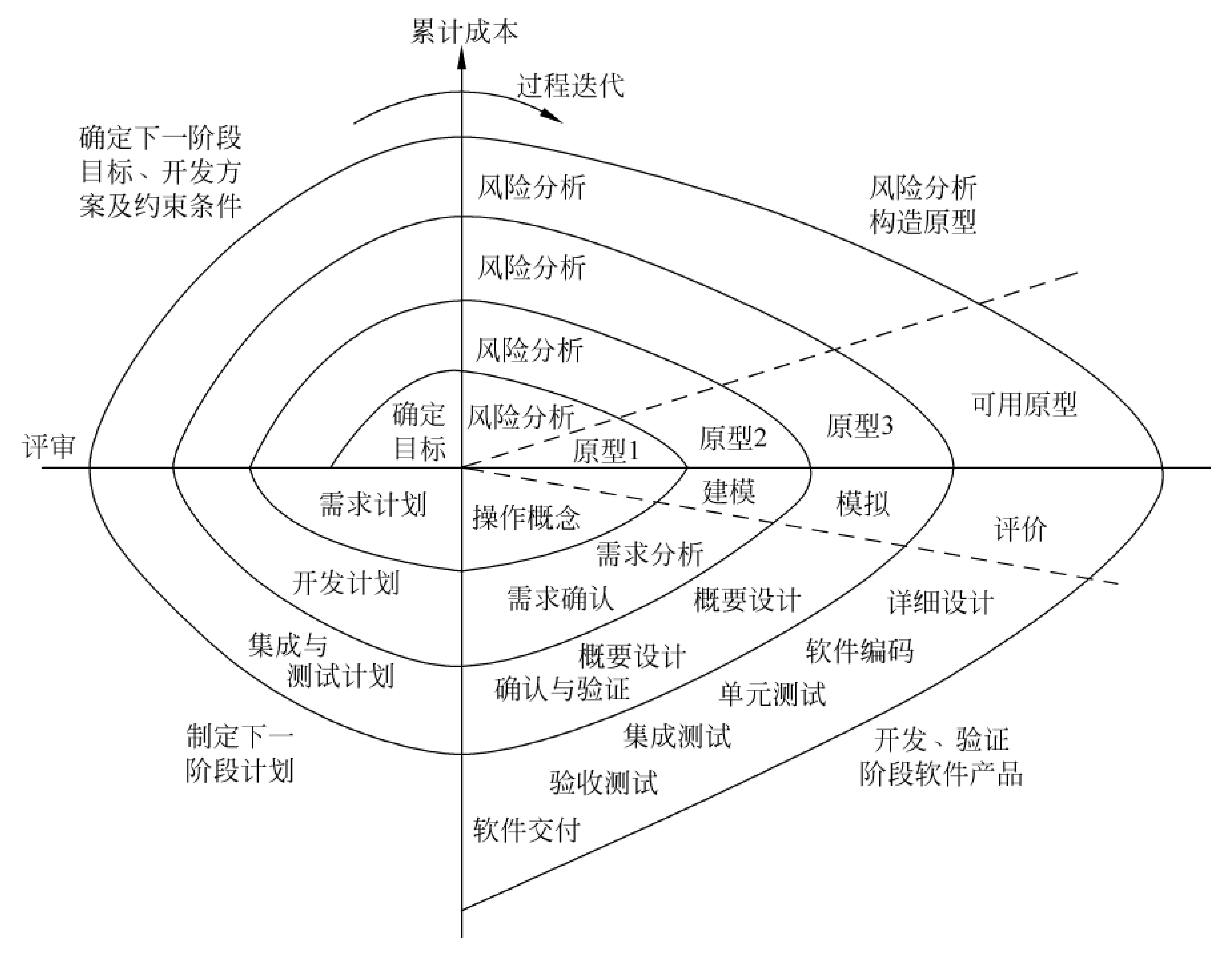


图 2.6 螺旋模型

在第一个螺旋周期，确定目标，进行概念性的系统设计，通过风险分析制定消除风险的方法，初步开发原型 1，制定系统开发计划；在第二个螺旋周期，进一步明确系统目标、开发方案及约束条件，通过风险分析制定消除风险的方法，在原型 1 的基础上开发原型 2，进一步明确软件需求，进行需求确认，修改开发计划；在第三个螺旋周期，再进一步确认系统目标、开发方案及约束条件，进行风险分析，制定进一步消除风险的方法，在原型 2 的基础上开发原型 3，此时可以进行产品设计，再对设计进行验证和确认，然后制定集成测试计划；在第四个螺旋周期，软件开发方案、系统目标和约束条件得到确定，在风险分析的基础上开发具有实用价值的可操作性原型。此时可对产品进行详细设计，进行软件编码、单元测试、集成测试，最后进行验收测试。验收合格后交付用户使用，进入运行、维护阶段。

2. 螺旋模型对经常遇见的问题提供的解决方案

螺旋模型对经常遇见的问题提供的解决方案如表 2.3 所示。

表 2.3 螺旋模型对经常遇见的问题提供的解决方案

经常遇见的问题	螺旋模型提供的解决方案
用户需求不够充分	允许并鼓励用户反馈信息
沟通不明确	在项目早期就消除严重的曲解
刚性的体系	开发首先关注重要的业务和问题
主观臆断	通过测试和质量保证，作出客观的评估
潜在的不一致	在项目早期就发现不一致问题
糟糕的测试和质量保证	从第一次迭代就开始测试
采用瀑布法开发	在早期就找出并关注风险

3. 螺旋模型的优缺点和适用情况

螺旋模型利用原型模型作为降低风险的机制,在任何一次迭代中均应用原型方法;同时,在总体开发框架上又保留了瀑布模型中系统性、顺序性和“边开发,边评审”的特点,这种将二者融合在一起的迭代框架更真实地反映了客观世界。螺旋模型的优缺点和适用情况如表 2.4 所示。

表 2.4 螺旋模型的优缺点和适用情况

优点	设计上的灵活性,可以在项目的各个阶段进行变更; 以小的分段来构建大型系统,使成本计算变得简单容易; 用户始终参与每个阶段的开发,保证了项目的方向与可控性; 具有瀑布模型和原型模型二者的优点
缺点	采用螺旋模型需要丰富的风险评估经验和专门知识,在风险较大的项目开发中,如果未能及时标识风险,势必造成重大损失; 过多的迭代次数会增加开发成本,延迟提交时间
适用情况	对于高风险、需求不确定的大型软件项目,螺旋模型是一个理想的开发过程模型

2.4.5 增量模型

对瀑布模型的一个关键性改进,出现了增量模型。增量模型是首先构建部分系统,再逐渐增加功能或者性能,直至完成整个系统。增量模型降低了取得初始功能之前的成本,强调采用构建方法来控制更改需求的影响,提高了创建可操作软件系统的速度。增量模型综合了瀑布模型和原型模型,提倡以功能渐增方式开发软件。经验表明,增量模型在特大型项目和小型项目中同样适用。增量模型描述了为系统需求排定优先级然后分组实现的过程,每个后续版本都对先前版本增加了新功能。在生命周期的早期阶段(软件规划、需求分析),需要建立整个系统架构,这个架构应该具有较强的可集成性,后续的构件方式开发都是建立在这个架构之上。重复生命周期的后续阶段(设计、编码、测试),每重复一次实现一个增量。首先创建一组核心功能,或者是项目至关重要的最高优先级的系统,或者是能够降低风险的系统;随后基于核心功能反复扩展,逐步增加功能以提高性能。增量模型结构如图 2.7 所示。

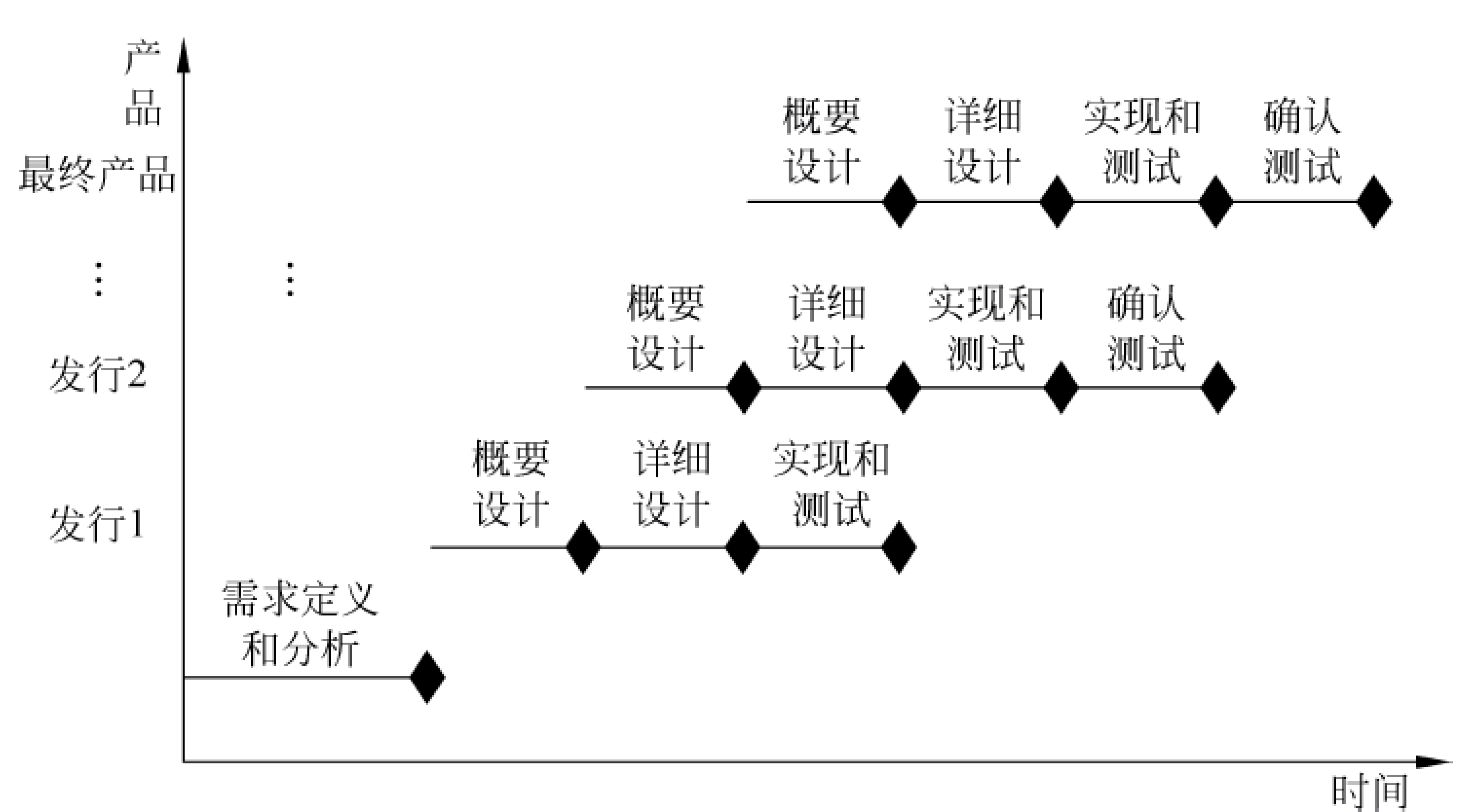


图 2.7 增量模型

1. 增量开发必须注意的问题

(1) 良好的可扩展性架构设计是增量开发成功的基础。

(2) 由于一些模块必须在另一个模块之前完成,所以必须定义良好的接口。

(3) 与完整系统相比,增量方式正式评审更难于实现,所以必须定义可行的过程。

(4) 要避免把难题往后推,首先完成的应该是高风险和重要的部分。

(5) 用户必须认识到总体成本不会更低。

(6) 分析阶段采用总体目标而不是完整的需求定义,可能不适应管理。

(7) 需要良好的计划和设计,管理人员必须注意动态分配工作,技术人员必须注意相关因素的变化。

2. 增量模型的优缺点和适用情况

增量模型的优缺点和适用情况如表 2.5 所示。

表 2.5 增量模型的优缺点和适用情况

优点	降低进度拖延、需求变更及验收问题的风险; 提高项目开发的可管理性; 采用连续增量的方式,把用户反馈融入到细化的产品中; 中间构件可以在最终版本完成之前交付,用户可以标识需求的变更; 采用"分而治之"的策略,将一个时间周期较长的项目分解开发; 在产品开发时,允许用户确认产品; 用户能够从早期的增量中了解系统,可以更改后面增量中的需求; 对尚不清楚的需求,可将实现推迟到弄清需求后的发行中
缺点	同瀑布模型一样,必须在早期就了解大部分需求; 对选择具体构件的开发方法敏感; 需要对每次发行进行回归测试,增加软件测试工作量; 生命周期的早期就将产品置于配置控制之下,因而需要正式的更改控制过程,将增加系统开销
适用情况	待开发项目类似于以前的成功项目; 大多数需求是稳定的和易于理解的; 整个项目开发时间大于一年,或者软件需要中期发行

2.4.6 RAD 模型

RAD(Rapid Application Development,快速应用开发)模型是增量型的软件开发过程模型,强调极短的开发周期,是瀑布模型的一个"高速"变种,通过大量使用可复用构件,采用基于构件的建造方法进行快速开发。RAD 模型结构如图 2.8 所示。

如果正确地理解了需求,而且约束了项目范围,利用该模型可以很快开发出功能完善的软件系统。流程从业务建模开始,随后是数据建模、过程建模、应用生成、测试交付。

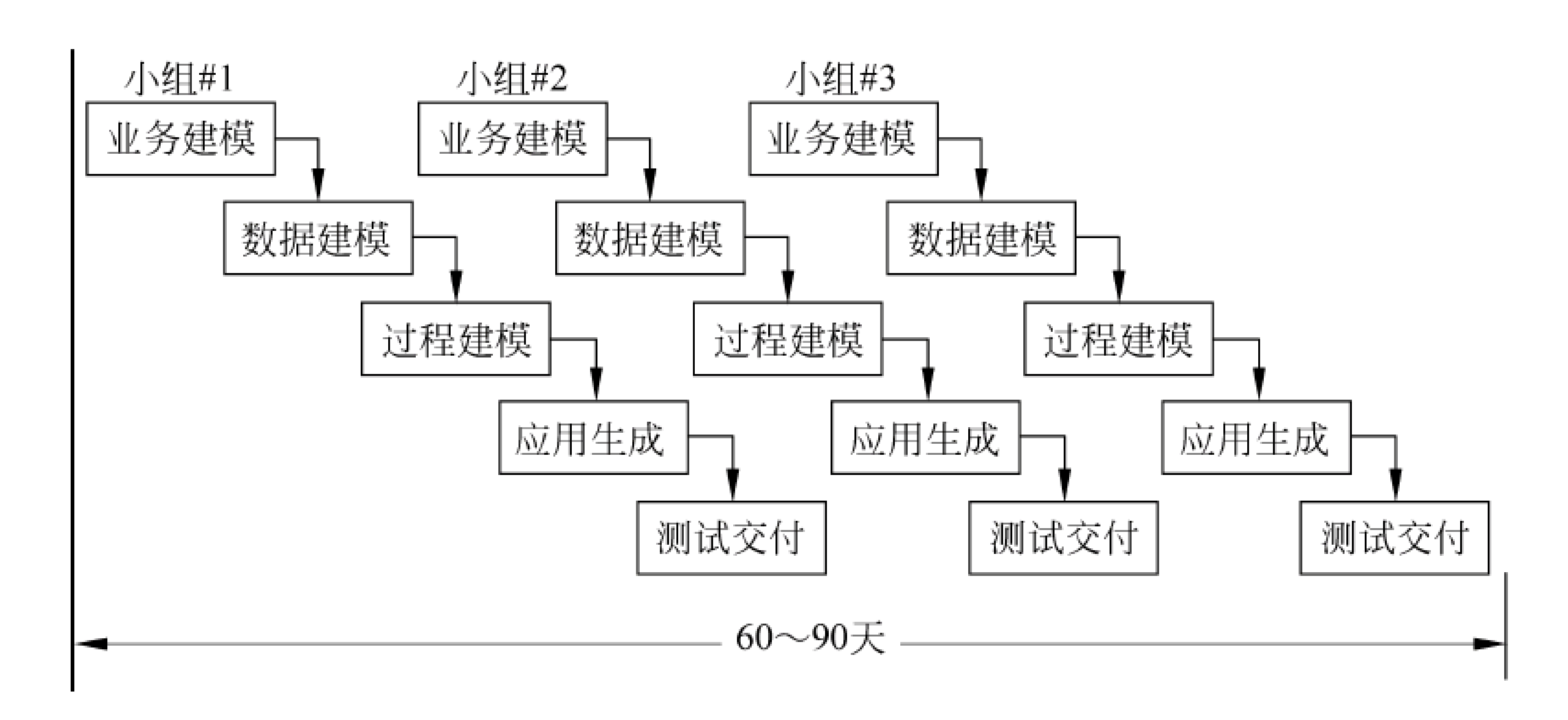


图 2.8 RAD 模型

1. RAD 模型各个活动期要完成的任务

与瀑布模型相比，RAD 模型不采用传统的第三代程序设计语言来创建软件，而是采用基于构件的开发方法，复用已有的程序结构，使用可复用构件，创建可复用构件，并使用自动化工具辅助软件开发。RAD 模型在项目时间上的约束需要“一个可伸缩的范围”。如果一个业务能够被模块化使得其中每一个主要功能均可以在不到三个月的时间内完成，则是 RAD 的一个候选。每一个主要功能可由一个单独的 RAD 组来实现，最后集成起来形成一个整体。RAD 模型各个阶段完成的任务如下：

(1) 业务建模。确定驱动业务过程运作的信息、欲生成的信息、如何生成、信息流的去向及其处理等，可以辅之以数据流图。

(2) 数据建模。为支持业务过程的数据流查找数据对象集合、定义数据对象属性，并与其他数据对象的关系构成数据模型，可辅之以 E-R 图。

(3) 过程建模。使数据对象在信息流中完成各业务功能，创建过程以描述数据对象的增加、修改、删除、查找，即细化数据流图中的处理。

(4) 应用生成，即应用程序生成。利用第四代语言(4GL)写出处理程序，重用已有构件或创建新的可重用构件，利用环境提供的工具自动生成以构造出整个应用系统。

(5) 测试交付，即包含测试与交付两个过程。由于大量重用，一般只做总体测试，但新创建的构件要进行其他测试。测试完成后进行系统集成，然后交付用户使用。

2. RAD 模型的缺陷

RAD 模型通过大量使用可复用构件加快了开发速度，对软件项目开发特别有效。但是与所有其他软件过程模型一样，RAD 模型也有缺陷，具体如下：

(1) 并非所有应用都适合 RAD。RAD 模型对模块化要求比较高，如果哪一个功能不能被模块化，那么建造 RAD 所需要的构件就会有问题。

(2) 开发人员和用户必须在很短的时间内完成一系列的需求分析，任何一方配合不当都会导致 RAD 项目失败。

(3) RAD 不适合技术风险很高的软件项目。当一个新应用要采用很多新技术，或当新软件要求与已有的计算机程序具有高互操作性时，技术风险较高，不宜采用 RAD。

2.4.7　软件包模型

软件包模型主要用于开发依赖于外购(协)软件产品和可重用软件包的系统。软件包模型结构如图 2.9 所示。

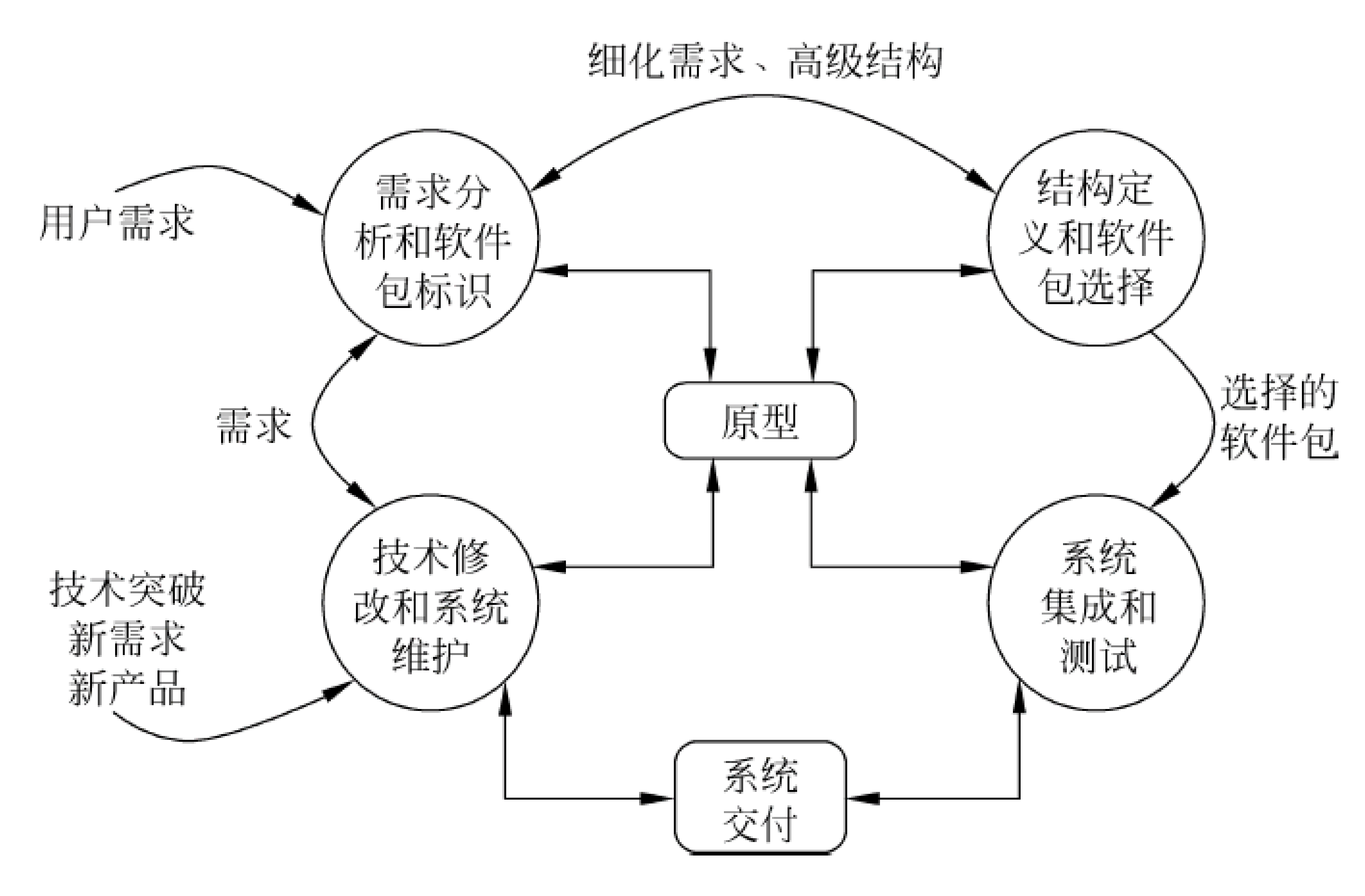


图 2.9　软件包模型

1. 软件包模型的开发步骤

软件包模型一般遵循下列开发步骤：

(1) 需求分析和软件包标识。在需求定义和分析期间，确定要使用的外购(协)软件包，构造原型系统以评价产品的功能和性能，并确定初步的系统结构。

(2) 结构定义和软件包选择。一旦原型的结果适合使用外购(协)软件，则应确定它与系统其余部分的接口，并确定系统的最终结构，否则选择其他外购(协)软件包，重新定义结构。

(3) 系统集成和测试。在实现期间，由于外购(协)软件包没有源代码，不能进行单元测试，因而直接进行系统集成和测试。

(4) 技术修改和系统维护。根据用户的使用情况，对交付后的系统进行技术修改和系统维护以改进系统。

2. 软件包模型的优缺点和适用情况

软件包模型的优缺点和适用情况如表 2.6 所示。

表 2.6　软件包模型的优缺点和适用情况

优点	与从头开发等价的功能相比，开发费用低； 与从头开发等价的功能相比，开发周期短； 可以提高最终产品的质量
缺点	可能会产生期望功能和外购软件提供功能之间的折中； 可维护性面临更大的挑战，因为外购软件的来源可能并不是同一开发机构(例如，外购(协)软件制造商发布更新版本时，需要第三方更改，并造成软件配置管理问题)
适用情况	外购软件可以提供待开发软件项目的大部分系统功能

2.4.8 遗留系统维护模型

遗留系统维护模型主要用于纠错性维护或者对一个运行系统稍加改进。如果需要改变软件结构，应使用瀑布模型或者增量模型。在维护期间，也可以执行某些在软件开发过程中经过选择的活动。遗留系统维护模型在本质上类似于瀑布模型，主要差别是该模型已经建立了结构设计。遗留系统维护模型结构如图 2.10 所示。

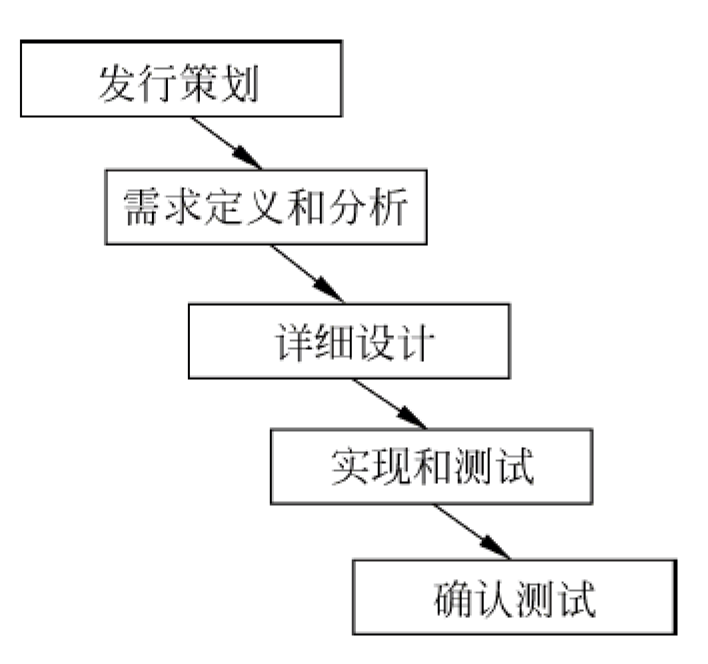


图 2.10 遗留系统维护模型

遗留系统维护模型的优缺点和适用情况如表 2.7 所示。

表 2.7 遗留系统维护模型的优缺点和适用情况

优点	定义清楚，易于建模和理解，便于计划和管理； 有支持该模型的多种工具； 适用于一个运行系统的纠错性维护或局部改进
缺点	不适用于需要改变软件结构的适应性维护； 不适用于需要改变软件结构的完善性维护； 不适用于新软件的开发
适用情况	只包含纠错及少量改进的维护发行

2.5 软件开发过程模型选择

目前，大多数软件开发项目都采用瀑布模型作为规范化开发的基础，主要原因如下：

(1) 软件开发单位的软件工程工作尚处于初级阶段，软件开发人员和管理人员既缺乏经验，又无历史数据可供借鉴，因此，需要一种简单易行的组织方式。

(2) 结构化方法学是系统工程中最成熟的方法学，目前大多数软件开发都以结构化开发方法学为基础。在与结构化方法学相适应的软件开发过程模型中，瀑布模型最为简单实用，行之有效。

(3) 有关软件开发的现行国家标准和国家军用标准都是以瀑布模型为基础制定的。

随着计算机技术的迅猛发展，新型软件支持工具和环境的不断推出，软件开发单位在软件开发经验和数据方面的日益积累，软件开发人员业务素质的逐步提高，未来软件开发将会采用更为先进的开发过程模型和技术。因此，在开发软件项目时，首先应当选定适当的软件开发过程模型，然后按选定的模型开展管理和技术工作，选用相应的标准和工具。对于软件开发项目，选择开发过程模型时，一般应遵循下述原则：

(1) 开发过程模型应与软件项目的特点(例如软件的规模和复杂性)相适应。

(2) 开发过程模型应与采用的软件开发技术(例如结构化方法)相适应。

(3) 开发过程模型应满足整个应用系统的开发进度要求。

(4) 开发过程模型应有助于控制和消除软件开发风险。

（5）开发过程模型应有可用的计算机辅助工具的支持。

（6）开发过程模型应与用户和软件开发人员的知识和技能水平相适应。

（7）开发过程模型应有利于软件开发的管理和控制。

在为一个具体项目选择开发过程模型时，通常应考虑项目的特点（如系统的功能和复杂性、软件的规模和复杂性、需求的稳定性、以前开发结果的使用、开发策略和硬件的可用性等），通过选择每一个过程的活动、规定活动的顺序和分配给活动的责任来定义软件开发过程。一个项目可以选择一个或多个软件开发过程模型。

思考题

1. 典型的软件生命周期包括哪些阶段？
2. 如何理解软件过程的定义及其软件过程管理的必要性？
3. 如何理解软件开发过程各阶段的主要任务？
4. 理论瀑布模型与实际瀑布模型有什么不同？
5. 瀑布模型的优缺点和适用情况是什么？
6. 试对 V 模型的水平和垂直关联进行比较分析。
7. 简述原型模型的基本过程。
8. 原型模型需要怎样的软件支撑环境？
9. 理解螺旋模型对经常遇见的问题提供的解决方案。
10. 软件增量开发需要注意哪些问题？
11. 为什么大多数软件开发项目都采用瀑布模型作为规范化开发的基础？
12. 选择软件开发过程模型时，一般应遵循哪些原则？

第3章 可行性研究

在项目的投资分析与决策过程中，可行性研究具有非常重要的地位和作用。这是因为作为一个具有百余年历史的科学方法，可行性研究是项目投资前期的一项重要内容，是项目投资决策过程中一个必不可少的程序，为项目投资决策提供直接的依据。本章主要讲述可行性研究的内容、可行性研究的阶段以及可行性研究的方法和技术。

3.1 可行性研究的含义

可行性研究(feasibility study)是在项目投资决策前，对拟建项目进行全面的技术经济分析与论证，并对其做出可行或不可行评价的一种科学方法。它是项目投资前期工作的重要内容，是项目投资决策中必不可少的一个工作程序。

在项目投资分析与决策过程中，可行性研究具体是指在项目投资决策之前，调查、研究与拟建项目有关的自然、社会、经济、技术等资料，分析、比较可能的项目投资建设方案，预测、评价项目建成后的社会经济效益，并在此基础上，综合论证项目投资建设的必要性、财务上的赢利性、经济上的合理性、技术上的先进性和适用性、建设条件上的可能性和可行性，从而为投资决策提供科学依据。

可行性分析可以看做是一次简略的系统分析和设计过程。所谓简略，就是不必深入到底层数据元素，也无须考虑每个处理细节。在简略的分析和设计基础上得到的系统方案，才能成为进行技术、经济和社会可行性分析的对象。

可行性研究报告是可行性研究阶段的主要成果，一个完整的可行性研究报告至少应包括以下三方面的内容：

(1) 分析论证项目投资建设的必要性。这一工作通常是通过市场预测工作(即通过市场预测，分析项目所生产产品的市场需求情况)来完成。软件项目则主要是分析项目给企业带来的各种益处。

(2) 分析项目投资建设的可行性。这一工作主要是通过技术分析和生产工艺来完成。软件项目主要是从技术角度、开发过程与方法等方面分析。

(3) 分析项目投资建设的合理性(财务上的赢利性和经济上的合理性)。这一工作主要是通过项目的效益分析来完成。项目投资建设的合理性是可行性研究的核心问题。

项目可行性研究的任务就是通过对拟建项目进行投资方案规划、工程技术论证、经济效益的预测和分析，经过多个方案的比较和评价，为项目决策提供可靠的依据和可行的建设方

案,并明确回答项目是否应该投资和怎样投资。

3.2 可行性研究的内容

3.2.1 技术可行性

技术可行性(technical feasibility)是可行性研究的重要内容,是分析在特定条件下,技术资源的可用性和这些技术资源用于解决软件项目问题的可能性和现实性。进行技术可行性研究首先要分析现有系统,绘制现有系统的系统流程图和高层数据流图,然后绘制所要开发新系统的系统流程图和高层数据流图。

将新系统与现有系统进行比较,分析所建议的系统可能带来的影响及其优越性,最后评价系统的技术可行性,即在限定的条件下,利用现有的技术,现有数量和质量的开发人员,在规定的期限内,开发是否能够完成。

进行技术可行性分析时,要注意以下一些问题。

1. 全面考虑系统开发过程所涉及的所有技术问题

软件开发涉及多方面的技术,包括开发方法、软硬件平台、网络结构、系统布局和结构、输入输出技术、系统相关技术等。应该全面和客观地分析软件开发所涉及的技术,以及这些技术的成熟度和现实性。

2. 尽可能采用成熟技术

成熟技术是被多人采用并被反复证明行之有效的技术,因此采用成熟技术一般具有较高的成功率。另外,成熟技术经过长时间、大范围使用、补充和优化,其精细程度、优化程度、可操作性、经济性要比新技术好。鉴于以上原因,在软件项目开发过程中,在可以满足系统开发需要、能够适应系统发展、保证开发成本的条件下,应该尽量采用成熟技术。

3. 慎重引入先进技术

在软件项目开发过程中,有时为了解决系统的特定问题,为了使所开发系统具有更好的适应性,需要采用某些先进或前沿技术。在选用先进技术时,需要全面分析所选技术的成熟程度。有许多报道的先进技术或科研成果实际上仍处在实验室阶段,其实用性和适应性并没有得到完全解决,也没有经过大量实践验证,在选择这种技术时必须慎重。例如,许多文章中已经报道了指纹识别技术,而且市场上也有实验性产品,但指纹识别技术至今仍有许多重大技术难题没有突破,离具体应用仍有一定距离。因此,在项目开发中就要谨慎选用这种技术。如果不加分析,在项目中盲目采用指纹识别技术,应用时肯定会出现许多难以解决的具体问题。

4. 着眼于具体的开发环境和开发人员

许多技术总的来说可能是成熟和可行的,但是在开发队伍中如果没有人掌握这种技术,而且在项目组中又没有引进掌握这种技术的人员,那么这种技术对本系统的开发仍然是不可行的。例如,分布对象技术是分布式系统的一种通用技术,但是如果在开发队伍中没有人

掌握这种技术，那么从技术可行性来看就是不可行的。

5. 技术可行性评价

技术可行性评价是通过原有系统和欲开发系统的系统流程图和数据流图，对系统进行比较，分析新系统具有的优越性，以及对设备、现有软件、用户、系统运行、开发环境、运行环境和经费支出的影响，然后评价新系统的技术可行性。技术可行性评价主要包括以下几个方面：

(1) 在限制条件下，功能目标是否能达到。

(2) 利用现有技术，性能目标是否能达到。

(3) 对开发人员数量和质量的要求，并说明能否满足。

(4) 在规定期限内，开发是否能够完成。

3.2.2 经济可行性

经济可行性(economic feasibility)分析也叫投资/效益分析或成本/效益分析，是分析开发软件项目所需要的花费，以及项目开发成功后所能带来的经济效益。通俗地讲，分析软件项目的经济可行性就是分析软件项目从经济角度是否值得开发。在可行性分析中，经济可行性是非常重要的。企业追求的目的就是效益和利润，如果收益小于支出，企业通常不会做这种亏本生意。

投资/效益分析需要确定所要开发软件项目的总成本和总效益，然后对总成本和总效益进行比较，当总效益大于总成本时，这个项目才值得开发。

软件项目总成本包括开发成本和运行成本。开发成本是指从立项到投入运行所花费的所有费用；而运行成本则是指软件项目投入使用后，系统运行、管理和维护所花费的所有费用。例如，某企业开发一个 ERP 系统，需要立项、可行性研究、需求分析、概要设计、详细设计、编码、测试、维护等过程，还要进行人员培训。ERP 系统运行后，为保证日常运行，还需要管理、操作和维护费用。ERP 系统每年的运行成本可能很低，但要年年支付，而且随着运行时间的延长，每年的维护工作量和成本将逐步增多，所有累计的运行成本并不一定比开发成本低。

软件项目总效益包括直接经济效益和间接社会效益。直接经济效益是软件项目能够直接获取的、并且能够用资金度量的效益。比如，降低的成本、提高的资金周转率、减少的人员成本以及减少的消耗等都是软件项目的直接经济效益，这些可以用资金进行计算。间接社会效益是指能够整体提高企业的信誉和形象，提高企业管理水平，但不能简单地或无法用资金计算的那部分效益。间接社会效益常常需要系统分析员根据本企业的状况和不同企业之间的类比进行概括估计。

通过比较成本和效益，就可以决定将要立项的软件项目是不是值得开发。一般情况下，比较的结论有三个：

(1) 效益大于成本，开发对企业有价值。

(2) 成本大于效益，不值得开发。

(3) 效益和成本基本持平，是否开发有待商榷。

在进行成本/效益分析时不要忽视软件项目给企业所带来的间接社会效益。简单地从

经济角度看，有些软件项目可能投入成本大于直接效益，但是它对企业带来的间接效益很大，这类系统仍然可以立项开发。

3.2.3　社会可行性

社会可行性（society feasibility）研究具有比较广泛的内容，它需要从政策、法律、道德、制度等社会因素论证软件项目开发的可行性和现实性。社会可行性主要是运行环境可行性和法律可行性。

1. 运行环境可行性

软件项目可行性分析不同于一般项目，软件项目产品大多数是一套需要安装并运行在用户单位的软件、相关说明文档、管理与运行规程等。只有软件正常使用，并达到预期的技术（功能、性能）指标、经济效益和社会效益指标，才能称为软件项目开发是成功的。

而运行环境是制约软件在用户单位发挥效益的关键。因此，需要从用户单位的管理体制、管理方法、规章制度、工作习惯、人员素质（甚至包括人员的心理承受能力、接受新知识和技能的积极性等）、数据资源积累、硬件和系统软件平台等多方面进行评估，以确定软件系统交付以后是否能够在用户单位顺利运行。

运行环境可行性分析中最重要的是操作可行性（operational feasibility）。操作可行性是指分析和测定软件系统在确定环境中能够有效地从事工作并被用户使用的能力和程度。操作可行性主要考虑以下方面：

（1）问题域的手工业务流程和新系统流程，两种流程的相近程度和差距。

（2）系统业务的专业化程度。

（3）系统对用户的使用要求。

（4）系统界面的友好程度以及操作的方便程度。

（5）用户的实际能力。

但在实际项目中，软件运行环境往往是需要再建立的，这就为项目运行环境可行性分析带来不确定因素。因此，在进行运行环境可行性分析时，可以重点评估是否可以建立系统顺利运行所需要的环境，以及建立这个环境所需要进行的工作，以便将这些工作纳入项目计划之中。

2. 法律可行性

软件项目涉及合同责任、知识产权等法律方面的问题，特别是在系统开发和运行环境、平台和工具等方面，以及产品的功能和性能方面，往往存在一些软件版权问题，例如是否能够购置使用环境和工具版权。法律可行性分析要分析软件项目是否会侵犯他人、集体或国家利益，是否违反国家法律，进而分析应承担的法律责任，并确定法律上是否可行。

3.3　可行性研究的阶段

联合国工业发展组织（UNIDO）编写的《工业项目可行性研究手册》中，把投资前期的可行性研究工作分为四个阶段，即机会研究、初步可行性研究、详细可行性研究和项目评估决

策。软件项目可行性研究的工作阶段也可参考工业项目可行性研究，分为这四个阶段。

由于软件项目前期各研究工作阶段的性质、工作目标、工作要求及作用不同，因而工作时间与费用也各不相同，如表3.1所示。通常因为各阶段研究的内容由浅入深，故项目投资额度和成本估算精度要求由粗到细，研究工作量由小到大，研究的目标和作用逐步提升，因而研究工作时间和费用也随之逐渐增加。

表3.1 可行性研究各阶段工作的目的和要求

研究阶段	机会研究	初步可行性研究	详细可行性研究	项目评估决策
研究性质	项目设想	项目初选	项目准备	项目评估
研究目的和内容	鉴别投资方向，寻求投资机会(含地区、行业、资源和项目的机会研究)，选择项目，提出项目投资建议	对项目作初步评价，进行专题辅助研究，广泛分析、筛选方案，确定项目的初步可行性	对项目进行深入细致的技术经济论证，重点对项目的技术方案和经济效益进行分析评价，进行多方案比选，提出结论性意见	结合分析各种方案，对可行性研究报告进行全面审核与评估，分析判断可行性研究的可靠性和真实性
研究要求	编制项目建议书	编制初步可行性研究报告	编制可行性研究报告	提出项目评估报告
研究作用	为初步选择投资项目提供依据，批准后列入项目前期工作计划，作为国家对投资项目的初步决策	判定是否有必要进行下一步详细可行性研究，进一步判明建设项目的生命力	作为项目投资决策的基础和重要依据	为投资决策者提供最后决策依据，决定项目取舍和选择最佳投资方案
估算精度	±30%	±20%	±10%	±10%
研究费用(占总投资的比例)	0.2%～1.0%	0.25%～1.25%	大项目0.2%～1%，中小项目1%～3%	—
需要时间/月	1～2	2～3	4～6或更长	—

3.3.1 机会研究

机会研究的主要任务是对软件项目的投资方向和设想提出建议，即根据国民经济和信息技术发展的长远规划，行业、地区规划，经济建设方针，工作任务和技术经济政策，在一个确定的地区、企业或部门内部，结合资源情况、市场预测和规划布局等条件，选择软件开发项目，寻找最有利的投资机会。

机会研究可分为一般机会研究和项目机会研究。一般机会研究就是对某个指定地区、行业或部门鉴别各种软件投资机会，或是识别利用以某种软件开发工具或网络产品为基础的投资机会。此项研究一般由国家机构和公共机构进行，作为制定信息化发展计划的基础。在对这些投资机会做出最初鉴别之后，再进行项目机会研究，即将项目设想转变为概略的项目投资建议，以引起投资者的注意，使其做出投资响应，并从几个有投资机会的项目中迅速而经济地做出抉择。然后，编制项目建议书，为初步选择投资项目提供依据。经批准后，列入项目前期工作计划，作为对投资项目的初步决策。

这一阶段的研究工作比较粗略，通常是根据类似条件和背景的软件项目来估算投资额度与开发成本，初步分析软件开发效果，提供一个或一个以上可能进行软件开发的投资项目和投资方案。这一阶段所估算的投资额和开发成本的精确程度大约控制在±30%的范围，大中型软件项目的机会研究所需要的时间在1～2个月，所需费用约占投资总额的0.2%～1.0%。如果投资者对这个项目感兴趣，则可进行下一步的可行性研究工作。

3.3.2　初步可行性研究

项目建议书经有关部门审定同意后，对于投资规模较大、技术较为复杂的大中型软件项目，仅靠机会研究还不能决定取舍，在开展全面研究工作之前，往往需要先进行初步可行性研究，进一步判明软件项目的生命力。这一阶段的主要工作目标是：

(1) 分析机会研究的结论，并在参考详细资料的基础上做出初步投资估算。该阶段工作需要深入弄清项目的规模、技术、研发组织结构、研发进度等，进行经济效果评价，以判定是否有可能和有必要进行下一步的详细可行性研究。

(2) 对某些关键性问题进行专题辅助研究。例如，市场需求预测和竞争能力研究，软件开发采用的关键技术和支撑环境研究，软件测试方法及软件维护方式期限研究等。在广泛的方案分析比较论证后，对各类技术方案进行筛选，选择效益最佳方案，排除一些不利方案，缩小下一阶段的工作范围和工作量，尽量节省时间和费用。

(3) 鉴定项目的选择依据和标准，确定项目的初步可行性。根据初步可行性研究结果编制初步可行性研究报告，决定是否有必要继续进行研究。如通过对所获得资料的研究确定该项目设想不可行，则应立即停止工作。该阶段是项目的初选阶段，研究结果应做出是否投资的初步决定。

(4) 初步可行性研究是介于机会研究和可行性研究之间的中间阶段，其研究内容和结构与可行性研究的内容和结构基本相同，主要区别是所获得资料的详尽程度不同，研究的深度不同。这一阶段对开发和维护成本估算的精度要求一般控制在±20%的范围，投资估算可采用算法模型、专家判定、动态分析、组件度量、推理估算等方法，研究所需时间为2～3个月，所需费用约占投资总额的0.25%～1.25%。

3.3.3　详细可行性研究

详细可行性研究是软件项目可行性研究的基础，为项目决策提供技术、经济、社会、财务等方面的评价依据，为项目的具体实施（即安装、调试和运行）提供科学依据。因此，这一阶段是进行详细深入的技术经济分析和论证阶段，主要目标是：

(1) 深入研究有关软件产品的技术方案、研发过程、硬件资源、网络环境、操作系统、开发工具、支持软件、进度计划、资金筹措计划、组织管理机构和人员以及各种可能选择的技术方案，进行全面深入的技术经济分析和比较选择工作，并推荐一个可行的软件开发投资方案。

(2) 着重对软件总体投资方案进行企业财务收益、国民经济收益和社会收益的分析与评价，对软件项目投资方案进行多方案比较选择，确定一个能使项目开发费用和维护成本降到最低限度，以取得最佳经济效益和社会效益的开发方案。

(3) 确定软件项目投资的最终可行性和选择依据标准。对拟投资开发的软件项目提出结论性意见。详细可行性研究的结论可以推荐一个认为最好的开发方案,也可以提出可供选择的几个方案,说明各个方案的利弊和可能采取的措施,或者也可以提出"不可行"的结论。按照详细可行性研究结论编制出可行性研究报告,作为软件项目投资决策的基础和重要依据。

(4) 详细可行性研究是决定项目性质的阶段(定性阶段),是项目决策研究的关键环节,该阶段为下一步的软件需求分析与设计提供基础资料和依据。因此,在此阶段,要求软件开发费用和维护成本估算精度控制在±10%的范围;研究工作所花费的时间为4~6个月;所需要费用,大型项目占总投资的0.2%~1%,中小型项目占总投资的1%~3%。

3.3.4 项目评估决策

项目评估是由投资决策部门组织或授权给相应的咨询公司或有关专家,对软件项目的可行性研究报告进行的全面审核和再评价。这一阶段的主要任务是对拟投资软件项目的可行性研究报告提出评价意见,对项目投资的可行与否做出最终决策(取舍),确定出最佳的软件开发投资方案。通常,项目评估应在可行性研究报告的基础上进行。项目评估决策主要包括以下内容:

(1) 审核可行性研究报告中反映的各项情况是否属实。

(2) 分析可行性研究报告中各项指标计算是否正确,包括各种参数、基础数据、软件成本估算方法等。

(3) 从公司、国家和社会等方面综合分析、判断软件项目的经济效益和社会效益。

(4) 分析和判断项目可行性研究的可靠性、真实性和客观性,对项目做出取舍的最终投资决策。

(5) 写出项目评估报告。

3.4 成本/效益分析

成本/效益分析的目的是从经济角度评价开发一个新软件项目是否可行。成本/效益分析首先估算待开发软件项目的开发成本,然后与可能取得的效益(有形和无形的)进行比较和权衡。有形效益可以用货币的时间价值、投资回收期、净收入等指标进行度量。无形效益主要是从性质上、心理上进行衡量,很难直接进行量的比较。无形效益在某些情形下会转化成有形效益。

系统的经济效益等于因使用新系统而增加的收入,再加上使用新系统可以节省的运行费用。运行费用包括操作员人数、工作时间、消耗物资等。

成本/效益分析用到的主要方法和技术有资金的时间价值、投资回收期、投资回收率等。有关资金的时间价值请参考相关的经济学书籍,在此不进行讲述。

3.4.1 投资回收期

投资回收期又叫投资返本期或投资偿还期,是指以项目净收益抵偿全部投资所需要的时间。这里所说的净收益主要是指利润,此外还可包括按制度规定允许作为还款用的折旧、

摊销及其他资金；全部投资包括固定资产投资、缴纳税款、借款利息等。

投资回收期是反映项目财务上投资回收能力的重要指标，是用来考察项目投资赢利水平的经济效益指标。计算投资回收期(以年为计算单位)一般从建设开始年算起，按是否考虑时间价值而分为静态投资回收期和动态投资回收期。

1. 静态投资回收期

静态投资回收期就是用项目各年的净收入将全部投资回收所需要的年限。这是最常使用的评价指标，具有直观、简单的特点，也能反映项目风险程度，但没有考虑资金的时间价值。

静态投资回收期的计算公式为：

$$\sum_{t=0}^{P_t}(\mathrm{CI}-\mathrm{CO})_t=0 \tag{3.1}$$

如果投产后的每年净收益相等，或用年平均净收益计算时，静态投资回收期的计算公式转化为：

$$P_t=\frac{K}{\mathrm{CI}-\mathrm{CO}}=\frac{K}{M} \tag{3.2}$$

式中：P_t——投资回收期(年)；

K——全部投资；

CI——现金流入量；

CO——现金流出量；

$(\mathrm{CI}-\mathrm{CO})_t$——第 t 年的净现金流量；

M——等额净收益或年平均净收益。

投资回收期亦可根据全部投资财务现金流量表中累计净现金流量计算求得，表中累计净现金流量等于零或出现正值的年份即为项目投资回收的终止年份。

设基准投资回收期为 P_c，判别准则为：

若 $P_t \leqslant P_c$，则项目可以接受；若 $P_t > P_c$，则项目应予以拒绝。

例如，某软件项目有 A、B 两个设计方案，各方案的现金流量如表 3.2 所示，计算静态投资回收期。

表 3.2 A、B 两个方案的静态投资回收期计算表 单位：万元

年份	方案 A			方案 B		
	投资	净收益	累计净现金流量	投资	净收益	累计净现金流量
1	200		−200	240		−240
2	100	60	−240		60	−180
3		70	−170		80	−100
4		80	−90		100	0
5		90	0		110	110
6		110	110		130	240

解：

方法一：利用公式(3.1)计算。

A 方案的投资回收期 P_{tA}：

$$\sum_{t=0}^{P_t}(CI-CO)_t = -200-40+70+80+90=0 \quad 得到\ P_{tA}=5(年)$$

B 方案的投资回收期 P_{tB}：

$$\sum_{t=0}^{P_t}(CI-CO)_t = -240+60+80+100=0 \quad 得到\ P_{tB}=4(年)$$

方法二：在现金流量表中计算。

在表 3.2 中，累计净现金流量等于零或出现正值的年份，方案 A 为 5，方案 B 为 4。

可见，两种计算方法得出的结果一致。

2. 动态投资回收期

为了克服静态投资回收期未考虑资金时间价值的缺陷，可采用动态投资回收期指标对方案进行评价和选择。

动态投资回收期是指在考虑资金时间价值条件下，按设定的利率回收全部投资所需要的时间。

动态投资回收期的计算公式为：

$$\sum_{t=0}^{P_t}\frac{(CI-CO)_t}{(1+i)^t}=0 \tag{3.3}$$

式中：i——年利率。

动态投资回收期同样也可根据全部投资财务现金流量表中累计净现金流量计算求得，表中累计净现金流量等于零或出现正值的年份即为项目投资回收的终止年份。

设基准投资回收期为 P_c，判别准则为：

若 $P_t \leqslant P_c$，则项目可以接受；若 $P_t > P_c$，则项目应予以拒绝。

在上例中，假定利率为 10%，计算动态投资回收期结果如表 3.3 所示。

表 3.3　A、B 两个方案的动态投资回收期计算表　　单位：万元

年份	方案 A				方案 B			
	投资	净收益	年净现金流量	累计净现金流量	投资	净收益	年净现金流量	累计净现金流量
1	200		−200.00	−200.00	240		−240.00	−240.00
2	100	60	−36.36	−236.36		60	54.55	−185.45
3		70	57.85	−178.51		80	66.12	−119.33
4		80	60.11	−118.40		100	75.13	−44.20
5		90	61.47	−56.93		110	75.13	30.93
6		110	68.30	11.37		130	80.72	11.65

累计净现金流量等于零或出现正值的年份，方案 A 为 6，方案 B 为 5。

利用公式(3.3)计算结果同表格中的结果一致。

由于考虑资金时间价值计算的结果，动态投资回收期大于静态投资回收期。但在投资

回收期不长和利率不大的情况下,两种投资回收期差别不大,不致影响项目或方案的选择。因此,只有在静态投资回收期很长的情况下,才有必要进一步计算动态投资回收期。

投资回收期带有明确的经济意义,计算简单、直观,便于投资者衡量项目的风险能力,并能在一定程度上反映投资效益的优劣。项目决策面临着未来不确定因素的挑战,这种不确定性所带来的风险随着时间的推移而增加。为了减少风险,人们希望投资回收期越短越好,基准投资回收期就是使项目风险尽可能减小的时间界限。因此,作为能够反映一定经济性和风险性的投资回收期指标,在项目评价中具有独特的地位和作用,并被广泛用做项目评价的辅助性指标。

但是投资回收期也有局限性:

(1) 没有考虑计划投资的项目使用年限。

(2) 没有考虑投资回收期以后的收益。

因此,投资回收期作为评价依据时,有时会使决策失误,往往与其他指标结合使用,以弥补其不足。

3.4.2 投资收益率

投资收益率(return on investment)也称投资报酬率,是指项目达到设计生产能力后,一个正常年份的净收益额与项目总投资的比率。对生产周期内各年净收益额变化幅度较大的项目,则计算生产周期年平均净收益额与项目总投资的比率。投资收益率的经济含义是表明项目投产后单位投资所创造的净收益额,因此,它是财务赢利能力分析和考察项目投资赢利水平的重要指标。计算公式为:

$$R = \frac{M}{K} \times 100\% \tag{3.4}$$

式中:R——投资收益率;

K——项目总投资;

M——项目投产后正常年份的净收益额或年平均净收益额。

根据分析目的不同,M 可以是年利润总额或年平均利润总额,也可以是年利税总额或年平均利税总额等。

设 i 为基准投资收益率,则判别准则为:

若 $R \geqslant i$,项目可以考虑接受;若 $R < i$,项目应予以拒绝。

在实际工作中,根据分析的目的不同,主要有下列三种投资收益率指标。

1. 投资利润率

投资利润率是指项目投产后正常年份的利润总额或生产期年平均利润总额与项目总投资的比率。计算公式为:

$$投资利润率 = \frac{年利润总额或年平均利润总额}{项目总投资} \times 100\% \tag{3.5}$$

投资利润率可根据项目评价损益表中的数据计算求得,并与部门或行业的平均利润率相比较,以判明项目单位投资赢利能力是否达到本行业平均水平。

2. 投资利税率

投资利税率是指项目投产后正常年份的利税总额或生产期年平均利税总额与项目总投资的比率。计算公式为：

$$\text{投资利税率} = \frac{\text{年利税总额或年平均利税总额}}{\text{项目总投资}} \times 100\% \tag{3.6}$$

投资利税率可根据项目评价损益表中的数据计算求得，并与部门或行业的平均利税率相比较，以判明项目单位投资对国家累积的贡献水平是否达到本行业平均水平。

3. 资本金利润率

资本金利润率是指项目投产后正常年份的利润总额或生产期年平均利润总额与项目资本金的比率。计算公式为：

$$\text{资本金利润率} = \frac{\text{年利润总额或年平均利润总额}}{\text{资本金}} \times 100\% \tag{3.7}$$

资本金利润率是反映项目投入资本金赢利能力的重要指标。

一般来说，投资收益率与投资回收期指标互为倒数关系，即：

$$R = 1/P_t \quad \text{或} \quad P_t = 1/R \tag{3.8}$$

基准投资收益率和基准投资回收期是作为项目评价的基准判定数据。而平均投资利润率与平均投资利税率用来作为衡量项目的投资利润率和投资利税率是否达到或超过本行业平均水平的评价判定数据，只作为项目评价的参考数据，不作为项目投资利润率和投资利税率是否达到本行业最低要求的判定数据。

3.5 方案选择与决策

在可行性研究阶段，通常有多个备选方案。方案选择与决策过程就是依据大量的可行性研究资料，运用科学方法，进行正确的评估和判断，选出最优方案的过程。方案选择的重要性可以概括为：只有选择正确，项目才能成功，如果选择错误，项目可能失败；只有选择正确，才是最大的节约，如果选择错误，就是最大的浪费。所以，方案选择与决策是可行性研究阶段的重要一环，选择正确与否决定着项目的成败得失和效益的高低与好坏，决定着整个项目的发展前途与命运，也间接决定着软件项目应用单位与开发单位的未来发展。

由于软件项目的特殊性，不能仅仅考虑收益，还要考虑其他很多因素，因而选择与决策方法较少，很多时候是人为主观决策。常用的比较直观的决策方法有极限图法，有时也运用运筹学的一些方法。这里简单介绍运筹学的两种方法。

3.5.1 确定型决策

确定型决策是指只存在一种确定的自然状态，有两个或两个以上的方案，各方案在确定条件下的收益值或损失值都可以量化和计算。这是一种简单的决策方法，对各方案进行对比分析，从中选取收益值（或收益率）最大的方案，或损失值（损失率）最小的方案作为决策结果。例如，某软件开发公司欲开发一个新的软件产品，在拟定方案中提出了三个可行方案：

方案 A、方案 B、方案 C。该公司对未来的市场需求进行预测，结果是该产品出现销售好、销售中等、销售差三种情况。三种情况投资收益率的估算结果如表 3.4 所示。

表 3.4　各方案投资收益率的估算结果

方案	销售好	销售中等	销售差
A	15%	20%	9%
B	22%	14%	12%
C	30%	11%	7%

根据表 3.4 的估算结果，如果可以肯定未来市场需求状况是销售好，则应选择方案 C，因为方案 C 可以为企业带来 30%的投资收益率，高于方案 A 的 15%和方案 B 的 22%；如果是销售中等，应选择方案 A；如果是销售差，应选择方案 B。

3.5.2　非确定型决策

非确定型决策是指对未来可能发生的情况虽然有所了解，但又无法确定或无法估计发生概率情况下的决策。由于评价原则及决策者的心理、素质不同，得到的决策方案也不同。常见的非确定型决策方法有以下几种。

1. 最大收益值(率)法

最大收益值(率)法又称乐观法，是先求出各方案在各种自然状态下可能的最大收益值(率)，然后比较各方案的最大收益值(率)，从中选出最大收益值(率)对应的方案为决策方案，即“大中取大”。这种方法常为喜欢冒险、有超人直觉并有较强承受能力的决策者采用。

对于表 3.4 的数据，要先找出各方案在各种状态下的最大收益率，分别为：A 方案 20%、B 方案 22%、C 方案 30%。其中以 C 方案的投资收益率 30%为最大，故选择 C 方案为决策方案。

2. 最大最小收益值(率)法

最大最小收益值(率)法又称悲观法，在西方也称瓦尔德决策准则，主要过程是先求出各方案在各种自然状态下的最小收益值(率)，然后比较这几个最小收益值(率)，从中选出最大者，即“小中取大”，作为所选方案。这种方法常为保守稳健的决策者所采用，因为可以保证在各种可能的情况下收益不低于此值。

对于表 3.4 的数据，各方案在各种自然状态下的最小收益率分别为：A 方案 9%、B 方案 12%、C 方案 7%。其中以 B 方案的投资收益率 12%为最大，故选择 B 方案为决策方案。

3. 最小最大后悔值(率)法

最小最大后悔值(率)法最初为萨凡奇所用，在西方也称做萨凡奇决策准则，主要过程是，当有多个方案可供决策者选择时，应先估计出每个方案在各种状态下的收益值(率)。当某一状态出现时，各个方案的收益值(率)是不同的，其中收益值(率)最大的那个方案就是该状态下的最好方案。如果决策者当初采用其他方案，就会后悔。所采用方案的收益值(率)

与最大收益值(率)之间的差,就称为该方案的后悔值(率)。在决策时,应先计算出各方案在各种状态下的后悔值(率),然后找出各方案的最大后悔值(率),再从各方案的最大后悔值(率)中选出最小者,即"大中取小",作为选择方案。这种方法常为那些因失败而后悔的决策者采用,因为可以使决策者的后悔程度最小。

在表 3.4 中,每种状态下的最大收益率分别为:销售好时 30%、销售中等时 20%、销售差时 12%。根据此收益率,计算出每个方案在各种状态下的后悔率,并从各方案的后悔率中找出每个方案的最大后悔率,如表 3.5 所示。从"最大后悔率"栏中可以看出,最小后悔率为 8%,其对应的方案 B 为决策方案。

表 3.5　各方案的最大后悔率

方案	销售好	销售中等	销售差	最大后悔率
A	15%	0%	3%	15%
B	8%	6%	0%	8%
C	0%	9%	5%	9%

4. 乐观系数法

乐观系数法又称折中系数法,是一种指数平均法,是介于最小收益值(率)与最大收益值(率)之间的评选标准,但可赋予最大收益值(率)以较高的权重,加重了最大收益值(率)在决策标准中的作用。这种方法首先采用加权平均方法计算出各个方案的折中收益值(率),然后选择最大者对应的方案为决策方案。计算公式为:

$$R_i = a \times \max(A_i) + (1-a) \times \min(A_i) \tag{3.9}$$

式中:R_i——各方案的折中收益率;

$\max(A_i)$——最大收益值(率);

$\min(A_i)$——最小收益值(率);

a——乐观系数,介于 0 和 1 之间。

对于表 3.4 中的数据,取乐观系数 a 为 0.6,则各方案的折中收益率分别为:

方案 A 的折中收益率 = 20% × 0.6 + 9% × 0.4 = 15.6%

方案 B 的折中收益率 = 22% × 0.6 + 12% × 0.4 = 18%

方案 C 的折中收益率 = 30% × 0.6 + 7% × 0.4 = 20.8%

在乐观系数 a 为 0.6 时,方案 C 的折中收益率为 20.8%,高于方案 A 和方案 B,因此选择方案 C 为决策方案。

5. 完全平均法

完全平均法又称等概率法,是假定各自然状态出现的概率完全相等,然后用各方案在各自然状态下的收益值(率)与假定的概率相乘,求出各方案的收益值(率),再从中选出最大的收益值(率)所对应的方案为决策方案。

对于表 3.4 中的数据,每种状态发生的概率为 1/3,则各方案的期望收益率分别为:

方案 A 的期望收益率 = (15% + 20% + 9%)/3 = 14.7%

方案 B 的期望收益率 = (22% + 14% + 12%)/3 = 16%

方案 C 的期望收益率 = (30% + 11% + 7%)/3 = 16%

从计算结果可以看出,方案 B 和方案 C 的期望收益率相等,都为 16%,高于方案 A 的 14.7%,因此,选择方案 B 或方案 C 为决策方案。

3.6 可行性研究报告的描述方法

对现有系统和建议系统的描述和表达,是可行性研究报告的重要内容。一个系统可以用逻辑模型和物理模型表达,前者着眼于功能表达,后者着眼于数据在系统各个物理元素(设备、文档、程序模块等)之间流动状况的表达。在撰写文档过程中,一般使用数据流程图以及辅助的文字说明表格来表示系统的逻辑模型,而用系统流程图及其辅助方案说明表格来描述系统的物理模型。

本节的图符依据《GB/T 1526—1989 信息处理 数据流程图、程序流程图、系统流程图、程序网络图和系统资源图的文件编制符号及约定》而编写。

3.6.1 数据流图

1. 概念与作用

数据流图(Data Flow Diagram,DFD)是一种图形化技术,描绘信息流和数据从输入移动到输出的过程所经受的变换。数据流图是描绘系统的逻辑模型,图中没有任何物理元素,只是描绘信息在系统中的流动和处理情况。因为数据流图是逻辑系统的图形表示,即使不是专业的计算机技术人员也容易理解,所以是极好的通信工具。此外,设计数据流图只需要考虑系统必须完成的基本逻辑功能,完全不需要考虑如何具体实现这些功能,所以是以后进行软件设计很好的出发点。

数据流图可以作为交流信息的工具。分析员把对现有系统的认识或对目标系统的设想用数据流图描绘出来,供有关人员审查确认。由于在数据流图中通常仅仅使用四种基本符号,而且不包括任何有关物理实现的细节,因此,绝大多数用户都可以理解和评价它。从数据流图的基本目标出发,可以考虑在一张数据流图中包括多少个元素合适的问题。

数据流图的另一个主要用途是作为分析和设计的工具。在研究现有系统时常用系统流程图表达对系统的认识,这种描绘方法形象具体,比较容易验证它的正确性。但是开发软件项目的目标不是完全复制现有系统,而是创造一个具备完成相同或类似功能的新系统。用系统流程图描述一个系统时,系统功能和实现每个功能的具体方案是混在一起的。因此,希望以另一种方式进一步总结现有系统,这种方式应该着重描绘系统所完成的功能,而不是物理系统方案。数据流图是实现这个目标的极好手段。

当用数据流图辅助物理系统设计时,以图中不同处理的要求为指南,能够在数据流图上画出许多组自动化边界,每组自动化边界可能意味着一个不同的物理系统,因此可以根据系统的逻辑模型考虑系统的物理实现。

2. 绘制内容

数据流图表示求解某一问题的数据通路,同时规定了处理的主要阶段和所用的各种数

据媒体。数据流图的绘制内容包括以下几个方面：

(1) 指明数据存在的数据符号，这些数据符号也可指明该数据使用的媒体。

(2) 指明对数据执行处理的处理符号，这些符号也可指明该处理用到的机器功能。

(3) 指明几个处理和/或数据媒体之间数据流的流线符号。

(4) 指明便于读、写数据流图的特殊符号。

在处理符号的前后都应是数据符号，数据流图以数据符号开始和结束。对于现有系统，可以在调查的基础上归纳出对数据的处理情况；而对于候选系统方案，在可行性研究阶段只需画出较简略和抽象的高层次数据流图。

3. 常用符号

数据流图有四种基本符号：数据的源点/终点、处理、数据存储、数据流，如图 3.1 所示。

(1) 源点/终点。用正方形或正方体表示数据流的产生处和最终抵达处。通常可能是系统外的部门、人员或组织。源点/终点如图 3.1(a)所示。

(2) 处理，有时也称变换数据的处理或加工。用圆角矩形或圆形表示对数据进行的操作，它是数据变换的原因。处理并不一定是一个程序，一个处理框可以代表一系列程序、单个程序或程序的一个模块，甚至可以代表用穿孔机穿孔，可目视检查数据正确性等人工处理过程。处理如图 3.1(b)所示。

(3) 数据存储。用开口矩形或两条平行横线表示处在静止状态，需要暂时存储的数据。一个数据存储并不等同于一个文件，它可以表示一个文件、文件的一部分、数据库的元素或记录的一部分等。数据可存储在磁盘、磁带、主存、微缩胶片、穿孔卡片及其他任何介质上。数据存储如图 3.1(c)所示。

(4) 数据流。用箭头表示含有固定成分的动态数据及流动方向。数据流与程序流程图中用箭头表示的控制流有本质不同，千万不要混淆。在数据流图中不能表示分支或循环，数据流中应描绘所有可能的数据流向，而不能描绘出现某个数据流的条件。数据流如图 3.1(d)所示。

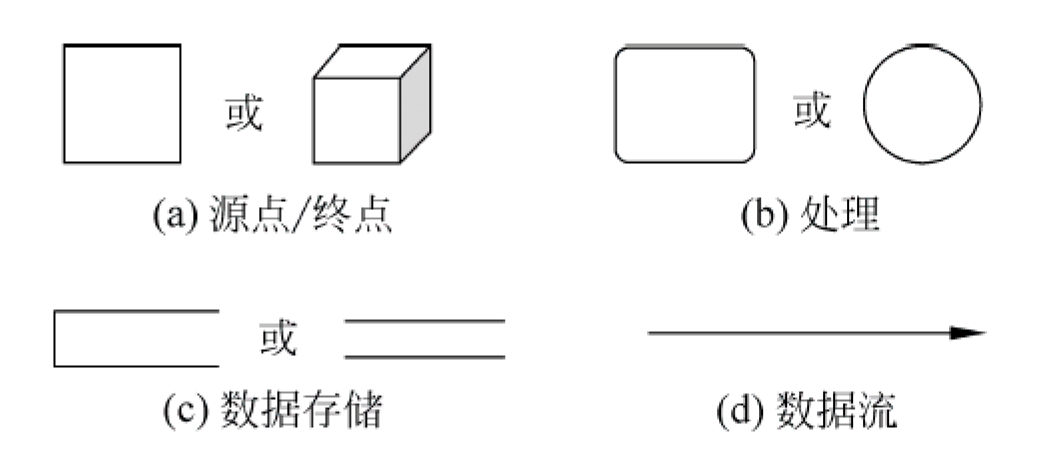


图 3.1　数据流图常用符号

另外需注意，数据存储和数据流都是数据，仅仅所处的状态不同。数据存储是处于静止状态的数据，数据流是处于运动状态的数据。

4. 实例

数据流图支持用结构化方法刻画系统功能。可以首先从系统角度描述系统的输入数据流和输出数据流以及对应的源点和终点，这是最高层次处理。然后，可以再进一步细分，正是在

这些处理的作用下，输入数据流才会逐步变换为输出数据流。

通常在数据流图中忽略出错处理，也不包括诸如打开或关闭文件之类的内部处理。数据流图的基本要点是描绘“做什么”，而不考虑“怎么做”。

有时数据的源点和终点相同，如果只用一个符号代表源点和终点，至少应有两个箭头和这个符号相连(一进一出)，可能其中一条箭头线相当长，这将降低数据流图的清晰度。另一种表示方法是重复画一个同样的符号表示数据的终点。有时数据存储也需要重复，增加数据流图的清晰程度。为了避免可能引起的误解，如果代表同一个事物的同样符号在图中出现 n 次，则在这个符号的一个角上画 $(n-1)$ 条短斜线做标记。

在处理较为复杂的系统时，通常采用分层方法。

下面以某信息传送管理系统完成信息制作及信息送达全过程为例，来说明数据流图的画法。

(1) 顶层数据流图。数据流图是系统的逻辑模型，任何计算机系统实质上都是信息处理系统，都是把输入数据变换成输出数据。因此，任何系统的顶层数据流图都是由若干个源点/终点以及一个处理组成，这个处理代表系统对数据加工变换的基本功能，处理的名称通常为系统名称。信息传送管理系统的顶层数据流图如图 3.2 所示。

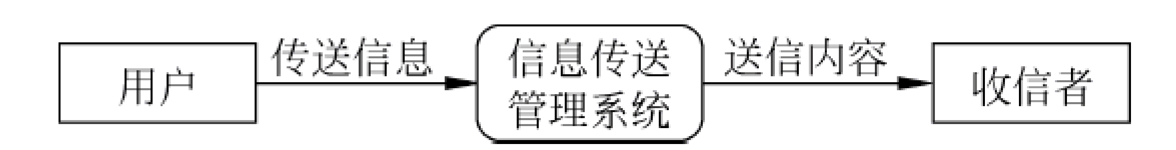


图 3.2 信息传送系统顶层数据流图

(2) 一层数据流图。一层数据流图也称为功能级数据流图。顶层数据流图过于抽象，对信息传送管理系统表述的信息非常有限。因此要将顶层数据流图细化，描绘系统的主要功能。系统的主要功能是信息传送制作和承认处理。细化后的数据流图还增加两个数据存储，信息传送制作需要送信内容信息表，承认处理需要送信状态信息表。一层数据流图如图 3.3 所示。为了便于引用和跟踪，对所有的处理和数据存储都进行编号。

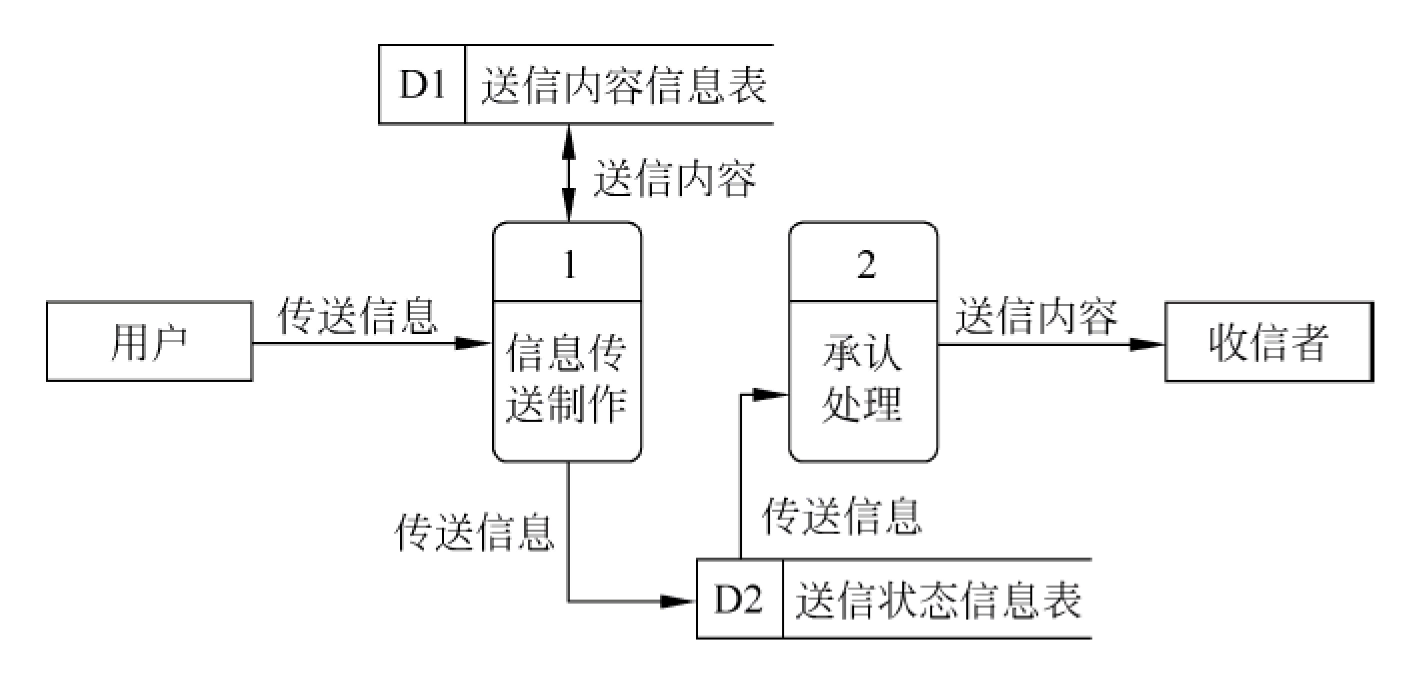


图 3.3 信息传送系统一层数据流图

(3) 二层数据流图。二层数据流图是对一层数据流图(功能级数据流图)中描绘的系统功能进一步细化。信息传送制作功能可进一步划分为送信部门制作、传送内容制作、传送对象设定等功能，如图 3.4 所示。当对数据流图细化时必须保持信息连续性，也就是说，当把一个处理分解为一系列处理时，分解前和分解后的输入输出数据流必须相同。在图 3.3 和图 3.4 中，输入数据流都是传送信息，输出数据流都是送信内容。

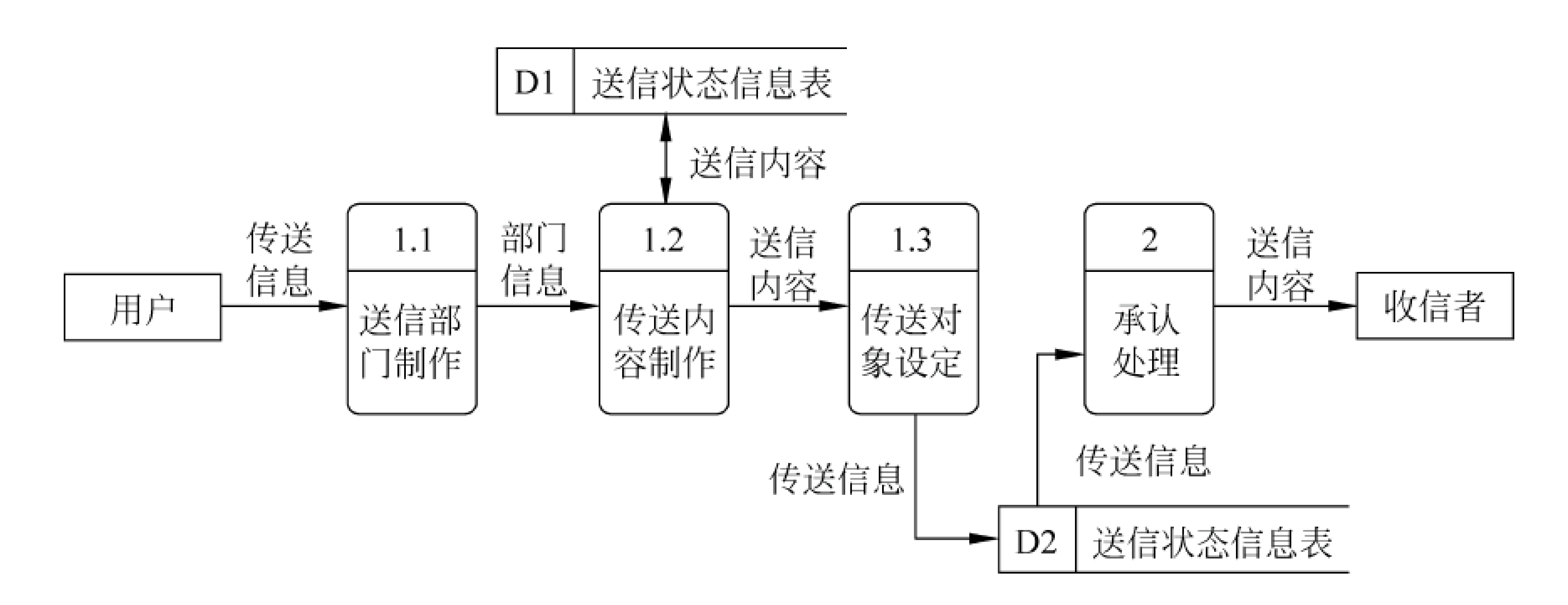


图 3.4 信息传送系统二层数据流图

另外，还要注意对处理进行编号的方法。处理 1.1、1.2、1.3 是更高层次的数据流图中处理 1 的组成元素；如果对处理 2 进一步分解，它的组成元素的编号应是 2.1、2.2、2.3、…；如果把处理 1.1 再进一步分解，则编号应为 1.1.1、1.1.2、1.1.3、…。

对二层数据流图是否还需要分解，根据具体情况而定。如果分解将涉及如何具体地实现一个功能时就不应该再分解了，否则应继续分解。

5. 命名

1）数据流和数据存储

(1) 名字代表整个数据流或数据存储的内容，而不仅仅反映它的某些部分。

(2) 不要使用空洞的、缺乏具体含义的名字，如“数据”、“信息”、“操作”等。

(3) 如果在为某个数据流或数据存储命名时遇到了困难，则很可能是因为对数据流图分解不恰当造成的，应该尝试重新分解，看是否能克服这个困难。

2）处理

(1) 通常先为数据流命名，然后再为与之相关联的处理命名，这样命名比较容易，而且体现了人类习惯的“由表及里”的思考过程。

(2) 名字应该反映整个处理功能，而不是它的一部分功能。

(3) 名字最好由一个具体的及物动词加上一个具体的宾语组成，尽量避免使用“加工”、“处理”等空洞笼统的动词作名字。

(4) 通常名字中仅包括一个动词，如果必须用两个动词才能描述整个处理功能，则把这个处理再分解成两个处理可能更恰当。

(5) 如果在为某个处理命名时遇到困难，则很可能是分解不当造成的，应考虑重新分解。

3）源点/终点

数据源点/终点并不需要在开发目标系统的过程中设计和实现，它不属于数据流图的核心内容，只不过是目标系统外围环境部分，如“人员”、“计算机外部设备”或“传感器装置”等。通常，为数据源点/终点命名时采用它们在问题中习惯使用的名称，如“仓库管理员”、“采购员”、“销售员”等。

3.6.2 数据字典

数据字典（Data Dictionary，DD）是关于数据信息的集合，也就是对数据流图中包含的

所有元素定义的集合。任何字典最主要的用途都是供人查阅不了解条目的解释，数据字典的作用也正是在软件分析和设计过程中，给人提供关于数据的描述信息。数据流图只能给出系统逻辑功能的一个总体框架而缺乏详细、具体内容。数据字典通过对数据流、数据元素、数据存储、加工等的描述，对数据流图的各种成分起到注解、说明作用，给这些成分赋予了实际内容。数据流图和数据字典共同构成系统的逻辑模型，没有数据字典的数据流图就不严格，没有数据流图的数据字典也难于发挥作用。只有数据流图和对数据流图精确定义的数据字典放在一起，才能共同构成系统的规格说明。

数据字典的主要用途是作为分析阶段的主要工具。在数据字典中建立一组严密一致的定义有助于改进分析员和用户之间的通信，因此将消除很多可能的误解。数据字典包含的每个数据元素的控制信息都是有价值的。因为列出了使用一个给定的数据元素的所有程序，很容易估计改变一个元素将产生的影响，并且能对所有受影响的程序或模块做出相应改变。

暂时还没有自动的数据字典处理程序，这里采用卡片形式书写数据字典，每张卡片上保存描述一个数据元素的信息。这种做法较好地实现了上述要求，特别是更新修改很方便，能够单独处理每个数据元素信息。每张卡片上包含的信息有名字、别名、描述、定义、位置等。

对于上面的数据流图，将数据字典的各条目分别举一个例子描述如下。

1. 源点/终点

名　　称：用户 别　　名：送信者或承认者 描　　述：具有登录信息、查询修改权限的用户 定　　义：用户＝用户 ID ＋密码＋姓名＋性别＋出生日期＋国家代码＋街道代码＋地址＋电话号码＋E-mail 地址 输入数据流：用户名、密码 输出数据流：送信内容 位　　置：用户信息表

2. 数据流

数据流名称：送信内容 别　　名：无 描　　述：对所要送信内容进行描述 数据流来源：送信内容制作 数据流流向：收信者 数据流组成：送信内容＝送信内容 No. ＋题名＋内容＋登录程序名＋登录日期＋登录者 ID＋登录者＋更新日期＋更新者 ID＋更新者 位　　置：送信内容信息表

3. 处理

名　　称：信息传送制作
加工编号：1
描　　述：在系统中制作欲传送信息及传送对象
输入数据流：送信内容
输出数据流：传送信息
处理逻辑：系统接收到登录用户传送信息要求，对所要传送信息的部门、送信内容、送信对象进行一系列有效制作，得到制作完成的传送信息，并且将得到的信息暂时保存，等待下一步处理

4. 数据存储

数据文件名：送信内容信息表
描　　述：存放送信内容信息
数据文件组成：送信内容信息表＝送信内容 No. ＋题名＋内容＋登录程序名＋登录日期＋登录者 ID ＋登录者＋更新日期＋更新者 ID ＋更新者
组织方式：按时间先后顺序排列
存取方式：顺序

思考题

1. 一个完整的可行性研究报告至少应包括哪些内容？
2. 简述可行性研究的内容。
3. 进行技术可行性分析时要注意哪些问题？
4. 具体说明经济可行性研究时的软件项目总成本和总效益。
5. 运行环境可行性中的操作可行性主要考虑哪些方面？
6. 详述可行性研究各阶段工作的目的和要求。
7. 初步可行性研究阶段的主要工作目标是什么？
8. 可行性研究阶段的主要工作目标是什么？
9. 项目评估决策阶段的主要内容包括哪些？
10. 简述投资回收期的定义。静态投资回收期和动态投资回收期有何区别？
11. 表明投资收益率有哪些指标？掌握各个指标的计算公式。
12. 简述确定型决策与非确定型决策的区别。
13. 非确定型决策具体有哪些方法？
14. 简述数据流图与数据字典的概念与作用。
15. 数据流图的绘制内容包括哪些？

第4章 需求分析

需求分析是软件生命周期的重要工作,也是决定性工作。只有通过需求分析,才能把软件功能和性能的总体概念描述为具体的软件需求规格说明,从而奠定软件开发的基础。

随着社会工业化、信息化、计算机及应用技术的迅速发展,各种各样的软件系统也随之出现,软件开发人员面临的应用系统复杂性越来越高,规模也越来越大,支持系统开发的基础软件面临着系统的复杂性、多样性、不间断性和自适应性等一系列关键问题的挑战。需求分析的重要性和必要性也就体现出来,需求分析质量的好坏直接影响整个软件工程的进展及最终结果。Bell 和 Thayer 指出:“不充分、相互矛盾以及非完全的软件需求描述是影响软件设计质量的一个非常关键的因素”,“设计一个系统的需求并不总是非常清楚,特别是对于较为复杂的系统,需要采用工程的观点,进行系统的分析和设计”。

4.1 需求分析概述

4.1.1 需求与需求分析

GB/T 11457—1995 软件工程术语中定义“需求”为:

(1) 用户为解决某一问题或达到某个目标所需要的条件或能力。

(2) 系统或系统部件为满足需要应具备的条件或能力,以满足合同、标准、规格说明或其他正式的强制性文件。所有需求的集合形成了以后开发系统或系统部件的基础。

在实际工作中,有时把需求细化为以下三个层次。

(1) 业务需求(business requirement):反映了组织机构或用户对系统、产品高层次的目标要求,在项目视图与范围文档中予以说明。

(2) 用户需求(user requirement):描述了用户使用产品必须要完成的任务,这在使用实例文档或方案脚本中予以说明。

(3) 功能需求(functional requirement):定义了开发人员必须实现的软件功能,使得用户能完成自己的工作,从而满足业务需求。软件需求各组成部分之间的关系如图 4.1 所示。

在软件需求规格说明(Software Requirements Specification,SRS)中的功能需求充分描述了软件系统应具有的外部行为。软件需求规格说明在开发、测试、质量保证、项目管理以及相关项目功能中都起重要作用。

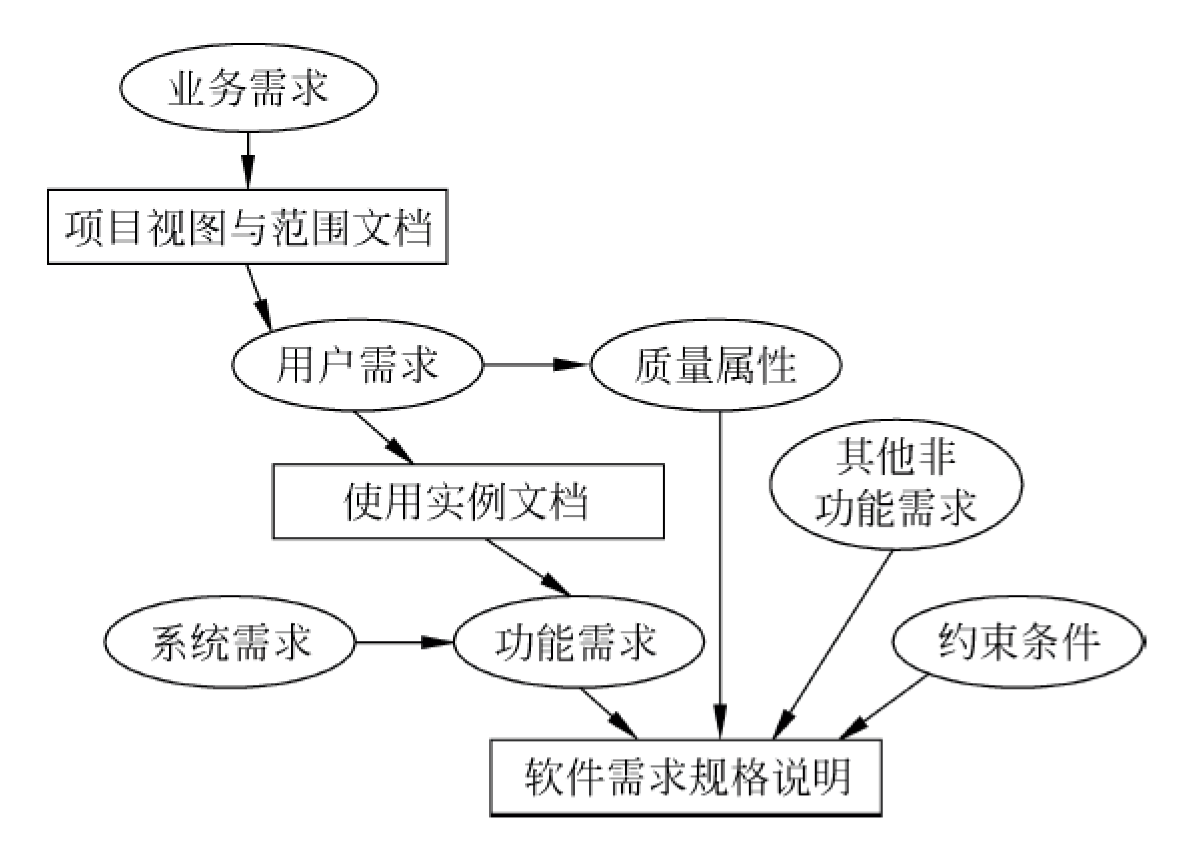


图 4.1 软件需求各组成部分之间的关系

作为功能需求的补充，软件需求规格说明还应包括非功能需求、约束条件、质量属性等。非功能需求描述系统展现给用户的行为和执行的操作，包括产品必须遵循的标准、规范和合约，外部界面的具体细节，性能要求；约束条件是指对开发人员在软件产品设计和构造上所具有的选择限制；质量属性是通过多种角度对产品的特点进行描述，从而反映产品功能，多角度描述产品对用户和开发人员都极为重要。

软件需求是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。通过对应问题及其环境的理解与分析，对问题涉及的信息、功能及系统行为建立模型，将用户需求精确化、完全化，最终形成需求规格说明，这一系列的活动即构成软件开发生命周期的需求分析阶段。

所谓“需求分析”，是指对要解决的问题进行详细分析，弄清楚问题的要求，包括需要输入什么数据、要得到什么结果、最后应输出什么。可以说，“需求分析”就是确定要计算机“做什么”。

在软件工程中，需求分析指的是在建立一个新的或改变一个现存的计算机系统时，描写新系统的目的、范围、定义和功能时要做的所有工作。需求分析是软件工程中的一个关键过程。在这个过程中，系统分析员和软件工程师确定用户的需求。只有确定了这些需求后才能够分析和寻求新系统的解决方法。

在软件工程的历史中，很长时间里人们一直认为需求分析是整个软件工程中最简单的一个步骤，但在过去十年中越来越多的人认识到它是整个过程中最关键的环节。如果在需求分析时分析者未能正确认识到用户需要，那么最后的软件实际上不可能达到用户需要，或者软件无法在规定的时间内完工。

4.1.2 需求分析的特点

需求分析是一项重要工作，也是最困难的工作。该阶段工作有以下特点：

(1) 用户与开发人员很难进行交流。在软件生命周期中，可行性研究阶段和需求分析阶段是面向用户的，其他阶段都是面向软件技术问题。需求分析是对用户的业务活动进行分析，明确在用户的业务环境中软件系统应该“做什么”。但是在开始时，开发人员和用户双方都不能准确地提出系统要“做什么”。因为软件开发人员不是用户问题领域的专家，不熟悉用户的业务活动和业务环境，不可能在短期内搞清楚；而用户又不熟悉计算机应用的有

关问题。由于双方互相不了解对方的工作,又缺乏共同语言,所以在交流时存在隔阂。

(2) 用户需求是动态变化的。对于大型复杂的软件系统,用户很难精确完整地提出功能和性能要求。开始时只能提出大概、模糊的功能,只有经过长时间的反复认识才逐步明确。有时进入到设计、编程阶段才能明确,更有甚者,到开发后期还在提新的要求。这无疑给软件开发带来困难。

(3) 系统变更的代价呈非线性增长。需求分析是软件开发的基础。假定在该阶段发现一个错误,解决它需要用一个单位的时间,到设计、编程、测试和维护阶段解决,则要花 2.5、5、25、50 倍的时间。

对于大型复杂系统而言,首先要进行可行性研究。开发人员对用户的要求及现实环境进行调查、了解,从技术、经济和社会因素等方面进行分析并论证软件项目的可行性,根据可行性研究的结果,决定项目的取舍。

4.1.3 需求分析的重要性

需求分析之所以重要,就是因为具有决策性、方向性、策略性的作用,在软件开发过程中具有举足轻重的地位。无论是在学习软件工程过程中,还是在软件开发实践中,一定要对需求分析具有足够的重视,它是开发出正确的、高质量软件的重要保证。需求分析在整个软件开发过程中的重要性主要表现在以下几点:

(1) 需求分析是获得用户需求的有效途径。开发软件是为用户服务的,为了开发出真正满足用户需求的软件产品,首先必须知道用户的需求。对软件需求的深入理解是软件开发工作获得成功的前提条件,不论软件开发工作者把设计和编码工作做得如何出色,不能真正满足用户需求的软件只会让用户失望。

(2) 需求分析是决定项目成功与否的关键因素。需求分析是项目的开端,也是项目建设的基石,在以往建设失败的项目中,有 80%是由于需求分析不明确造成的。因此,项目成功与否的关键因素之一就是对需求分析的把握程度,项目的整体风险具体表现在需求分析不明确、业务流程不合理上,用户不习惯或者不愿意使用开发出的软件,或者很难应用,从而造成了项目失败。

(3) 需求分析是系统分析和软件设计的桥梁。需求分析过程是确定用户需求的过程,用户知道自己的需要,却不懂得如何用计算机技术实现;而软件设计人员和程序员往往缺乏理解实际事务的运行过程和商业过程的技巧。那么通过专门训练的系统分析员就填补了商业领域和计算机世界之间的鸿沟。他们能够从持有关键信息的用户那里获得可用信息,并把这些用户信息转化为清晰完整的形式。这些能被软件工程师理解的形式就是进行下一阶段——软件设计的依据。

(4) 需求分析是控制软件质量的重要阶段。在软件生命周期的每个阶段都采用科学的管理方法和先进的技术手段,而且在每个阶段结束之前,都从技术和管理两个角度进行严格的审查,合格之后才开始下一个阶段的工作,这就使软件开发工程的全过程以有条不紊的方式进行,能够保证软件质量,特别是能提高软件的可维护性。需求分析阶段是项目的开始阶段,也是质量控制的开始。在需求分析阶段如果出现了问题,在后面阶段问题也随之留下来,并对后面阶段产生了“乘数效应”——影响变得越来越大。软件质量可能就是由这一步的失误、错误、粗心等因素造成的。

4.1.4 需求分析的任务

需求分析的任务不是确定系统怎样完成工作，而是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。需求分析的主要工作就是深入描述软件的功能和性能，确定软件设计的限制和软件同其他元素的接口细节，定义软件的有效性要求，研究用户要求，准确表达被接受的用户要求，确定被开发软件的系统元素，最后将功能和信息结构分配到这些系统元素中。因此，需求分析的任务就是借助于当前系统的逻辑模型，导出目标系统的逻辑模型，解决目标系统“做什么”的问题，如图 4.2 所示。

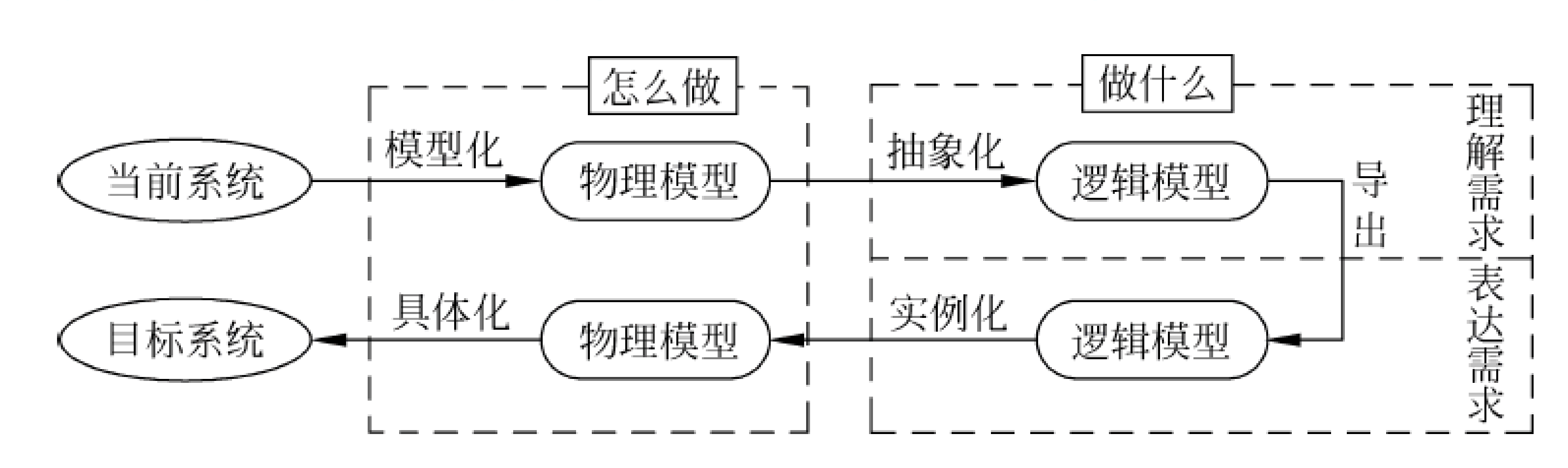


图 4.2 参考当前系统建立目标系统模型

由图 4.2 可知，需求分析的实现步骤大致是：

(1) 获得当前系统的物理模型。

(2) 抽象出当前系统的逻辑模型。

(3) 导出目标系统的逻辑模型。

(4) 最终目标系统的物理模型是由逻辑模型经实例化，即具体到某个业务领域而得到的。

4.2 需求分析过程

需求分析的过程是：获取用户需求→分析用户需求→编写需求文档→评审需求分析，如图 4.3 所示。如果评审通过，则需求分析结束；如果评审未通过，可能是由于编写需求文档、分析用户需求、获取用户需求等某个步骤引起的，要返回到相应的步骤进行修改。

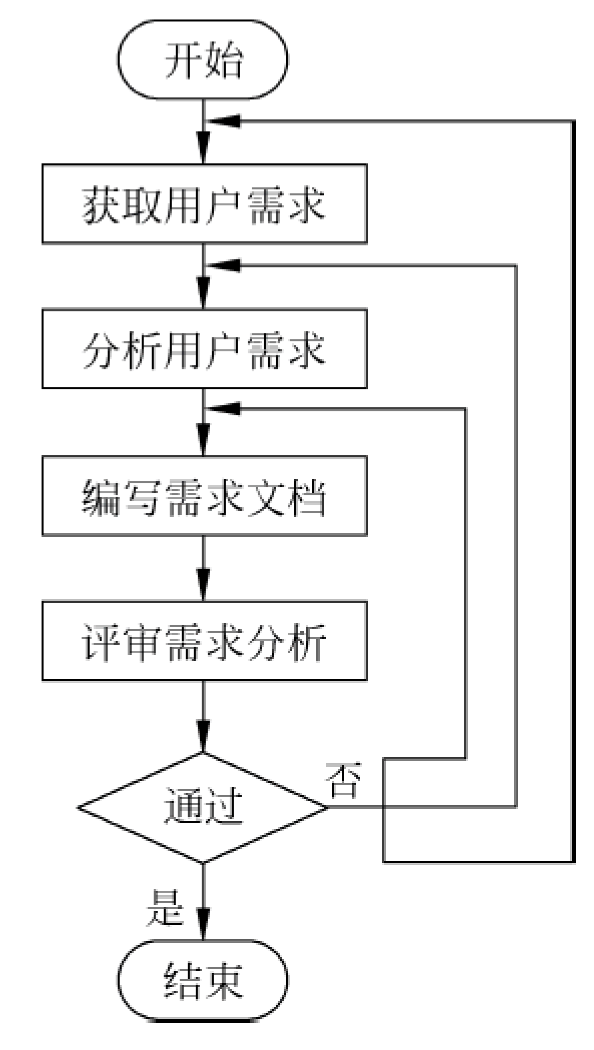


图 4.3 需求分析的过程

4.2.1 获取用户需求

按照软件工程对软件开发过程的描述，需求阶段必须充分细致地了解用户目标、业务内容、流程等，这是对需求的采集过程，是进行需求分析的基础准备。获取用户需求要了解所有用户类型及潜在类型，确定系统的整体目标和工作范围。获取用户需求要完成的主要工作如下：

(1) 依据分析阶段确定合适的用户方配合人员。在需求调研前，应该对用户方配合人员进行分类，使之和分析的各个阶段相对应。经过以下三个阶段，对需求的描述将比较准确和完整。

① 分析初期，即总体分析阶段。这一阶段需要得到整体意

义上的轮廓需求，应与用户方较高层次的人员进行交流，对未来的系统应有完整的描绘，可以划分出子系统及其之间的关系，这也是高级管理层对系统的期望。以此作为纲领性的文档指导进一步分析，并约束后续的分析过程，避免需求范围漫无边际地扩大。

② 专业系统分析阶段。由于用户内部业务上有分工，彼此之间既相互独立，又相互发生联系。此阶段应与用户方专业人员进行深入讨论。这一层次的人员对自己的专业熟悉，对专业内的需求到位，大都年富力强，有相当的阅历和理解能力，甚至可以绘制业务流图，总结业务功能点，要充分调动这些人员的积极性。

③ 系统关联分析阶段。在各专业系统得到充分分析的基础上，就要理清系统之间的关系，这是提升需求层次的关键阶段，也是高层领导和专业技术人员都关心的阶段。通常，用户单位都会有一些零散的软件，如财务软件、上级下发的软件等，这些专业软件都发挥着重要作用，但都是些信息孤岛，用户希望能把这些信息融合到整个系统中去，为更多的人共享。同时，也希望数据能够在各专业系统间顺畅流动，从而减少重复劳动，提高工作效率。此阶段应把专业技术人员召集到一起，共同商定系统间的接口。

(2) 多方位描述同一需求。有些需求贯穿了从基层人员到高层领导的需要，此类需求应该从各个角度、各个方位进行描述，综合之后才能得到完整的表达，否则可能会漏掉一些信息。这也为后续的设计工作打好了基础。

(3) 清晰化每一数据项。由于需求是设计的基础，弄清所有数据项的来龙去脉对于设计是必不可少的，不能有模糊不清的内容。同时，通过对数据项来源的分析，可以使分析人员更清楚地看到数据的流动情况，也会发现一些新的需求点。

(4) 充分挖掘潜在需求。由于分析人员对软件技术非常熟悉，一些由于技术所带来的潜在需求对于用户来说一般很难发现。不实现这些需求对整个系统也没什么实质性的影响，实现这些需求则会锦上添花。

对用户进行访谈和调研，交流方式可以是会议、电话、电子邮件、小组讨论、模拟演示等不同形式。需要注意的是，每次交流一定要有记录，对于交流的结果可以进行分类，便于后续的分析活动。

需求分析人员将调研的用户需求以适当的方式呈交给用户方和开发方的相关人员。大家共同确认需求分析人员提交的结果是否真实地反映了用户意图。需求分析人员在这个任务中需要执行下述活动：明确标识出那些未确定的需求项(在需求分析初期往往有很多这样的待定项)；使需求符合系统的整体目标；保证需求项之间的一致性，解决需求项之间可能存在的冲突。

4.2.2 分析用户需求

分析用户需求的主要工作是对问题的分析和方案的综合。分析员要从数据流和数据结构出发，逐步细化所有软件功能，找出系统各元素之间的联系、接口特性和设计上的限制，分析是否满足功能要求，是否合理。依据功能需求、性能需求和运行环境需求等，剔除不合理的部分，增加需要的部分。最终综合成系统的解决方案，给出目标系统的详细逻辑模型。

在这个步骤中，分析和综合工作反复进行。在对现行系统和期望的信息(输入和输出)进行分析的基础上，分析员往往综合出一个或几个解决方案，然后检查是否符合软件计划中规定的范围，再进行修改。总之，对问题进行分析和综合的过程将一直持续到分析员与用户

双方都感到有把握正确地制定软件的需求规格说明为止。

常用的需求分析方法有面向数据流的结构化分析方法(简称 SA)、面向数据结构的 Jackson 方法(简称 JSD)、面向对象的分析方法(简称 OOA)以及用于建立动态模型的状态迁移图或 Petri 网等。这些方法都采用图文结合的方式,可以直观地描述软件的逻辑结构。

4.2.3 编写需求文档

已经确定的需求应当得到清晰准确的描述。通常把描述需求的文档叫做软件需求规格说明书。同时,为了确切表达用户对软件的输入输出要求,还需要制定数据要求说明书及编写初步的用户手册,着重反映被开发软件的用户界面和用户使用的具体要求。

需求分析的最终结果是用户和开发小组对将要开发的产品达成一致协议。协议综合了业务需求、用户需求和软件功能需求。

1. 编写软件需求文档的方法

可以使用以下方法编写软件需求文档:

(1) 最好用结构化和自然语言编写文本型文档。

(2) 建立图形化模型,用以描绘转换过程、系统状态、状态的变化过程、数据关系、逻辑流、对象类、对象之间的关系等。

(3) 编写形式化规格说明,可以通过使用数学上精确的形式化逻辑语言来定义需求。

2. 编写软件需求文档要注意的问题

编写好的软件需求文档要注意以下问题:

(1) 需求说明要全面。用户往往只提功能性需求,对非功能性需求只字未提,但需求人员不能遗漏这部分内容。非功能性需求是指性能、安全性、可靠性、可维护性等,不影响用户的业务功能,但若系统运行不稳定,常常是由非功能性需求考虑不周到引起的。

(2) 功能细节的描述要准确,不能有歧义。开发人员和用户很少有面对面的交流机会,一个问题如果需求人员描述不清晰,文档提交给开发人员后,极易被理解成完全不同的意思。但有时这是难以避免的,只能在需求评审时进行过滤。

(3) 需求说明书要将用户的想法提炼成可实现的功能。要尽量忠实于用户的想法,即使用户需求在技术实施上有难度,也不能任意删减。可以在需求评审时向用户说明,并将替代方案介绍给用户,征得用户同意。要用诚信去赢得用户,如果需求中任意删减用户认为关键的功能,待用户以后发现时再次提起,容易造成不必要的麻烦。

4.2.4 需求分析评审

需求分析直接关系到软件产品的方向,所以需求分析的质量至关重要。对于这个里程碑的质量控制,可以通过内部评审和同行评审的方式,然后是用户评审。项目组内部评审或同行评审主要是根据公司规范和评审人员自身的经验,对需求分析中不明确、不合理、不符合逻辑、不符合规范的地方予以指正。用户评审主要是对描述的软件功能是否真正符合需求,能否帮助用户解决问题等方面做出评定,所以需求分析评审时用户意见是第一位的。

作为需求分析阶段工作的复查手段，在需求分析的最后一步，主要对功能的正确性、完整性和清晰性以及其他需求给予评审。评审的主要内容是：

- 系统定义的目标是否与用户的要求一致；
- 系统需求分析阶段提供的文档资料是否齐全；
- 文档中的所有描述是否完整、清晰，是否准确地反映了用户要求；
- 与所有其他系统成分的重要接口是否都已经描述；
- 所开发项目的数据流与数据结构是否完整；
- 所有图表是否清楚，在不补充说明的情况下能否理解；
- 主要功能是否已包括在规定的软件范围之内，是否都能充分说明；
- 设计的约束条件或限制条件是否符合实际；
- 开发的技术风险是什么；
- 是否考虑过软件需求的其他方案；
- 是否考虑过将来可能会提出的软件要求；
- 是否详细制定了检验标准，这些标准能否对系统定义成功地进行确认；
- 有没有遗漏、重复或不一致的地方；
- 用户是否审查了初步的用户手册；
- 软件成本进度估算是否受到了影响等。

4.3 需求分析内容

需求分析的内容具体包括以下几个方面：

(1) 功能需求。软件功能是软件应具有的效能和作用，软件目标要通过软件功能来表达和实现，软件功能也是软件呈现给用户的直接效果。用户通过软件提供的功能来认识、使用和评价系统，通过使用软件功能来完成业务工作。功能需求是需求分析最重要的内容。功能分析是依据软件目标，形成用软件功能模型描述的结果，并定量或定性地叙述对软件提出的功能要求。

(2) 性能需求。软件的技术性能指标，主要是精度、时间特性要求和灵活性。

① 精度。说明对软件输入、输出数据的精度要求，可能包括传输过程中的精度。

② 时间特性要求。说明对软件的时间特性要求，包括响应时间、更新处理时间、数据转换和传送时间、计算时间等。

③ 灵活性。说明对软件的灵活性要求，即当需求发生变化时，软件对变化的适应能力。包括操作方式的变化、运行环境的变化、同其他软件接口的变化、精度和有效时限的变化、计划的变化或改进等。为了提供这些灵活性而进行的专门设计部分应该标明。

(3) 输入输出要求。解释各输入输出数据类型，并逐项说明其媒体、格式、数值范围、精度等。对软件的数据输出和必须标明的控制输出量进行解释并举例，包括对硬拷贝报告(正常结果输出、状态输出及异常输出)以及图形或显示报告的描述。

(4) 数据管理能力要求。说明需要管理的文卷和记录的个数、表和文卷的大小规模，要按可预见的增长对数据及其分量的存储要求做出估算。

(5) 环境需求。软件系统运行时所处环境的要求，包括硬件方面的机型、外部设备、数

据通信接口，软件方面的操作系统、网络软件、数据库管理系统，使用方面的使用部门在制度上、操作人员在技术水平上应具备的条件。

(6) 可靠性要求。对软件投入运行后不发生故障的概率，按实际的运行环境提出要求。对于重要的软件，或是运行失效会造成严重后果的软件，应提出较高的可靠性要求。

(7) 安全保密要求。应当在这方面恰当地做出规定，对开发的软件给予特殊设计，使其在运行中安全保密方面的性能得到保证。

(8) 用户界面需求。为用户界面规定达到的要求。

(9) 资源使用需求。软件在运行时和开发时所需要的各种资源。

(10) 软件成本消耗与开发进度需求。软件项目立项后，根据合同规定，对软件开发的进度和各步骤的费用提出要求，作为开发管理的依据。

4.4 需求分析方法

需求分析方法由对软件问题的信息域和功能域的系统分析过程及其表示方法组成，大多数的需求分析方法由信息驱动。信息域有三种属性：信息流、信息内容和信息结构。

常用的需求分析方法有很多，选择哪种方法，要依据哪些资源在什么时间对开发人员有效，不能盲目套用。下面对常用的几种方法进行阐述。

4.4.1 结构化方法

结构化开发方法(Structured Developing Method)是现有的软件开发方法中最成熟、应用最广泛的方法，主要特点是快速、自然和方便。结构化开发方法由结构化分析方法(SA方法)、结构化设计方法(SD方法)及结构化编程方法(SP方法)构成。

结构化分析(Structured Analysis，SA)方法是面向数据流的需求分析方法，20世纪70年代末由Yourdon、Constaintine及DeMarco等人提出，并得到广泛的发展和应用。结构化分析方法适用于分析大型的数据处理系统，特别是管理信息系统。

1. 结构化分析的基本思想

SA方法也是一种建模活动，主要根据软件内部的数据传递、变换关系，自顶向下逐层分解，描绘出满足功能要求的软件模型。

SA方法的基本思想是"分解"和"抽象"。分解是指对于一个复杂系统，为了将复杂性降低到能掌控的程度，把大问题分解成若干小问题，然后分别解决；抽象是指用最本质的属性表示一个系统的方法，即先考虑问题最本质的属性，暂把细节略去，以后再逐层添加细节，直至涉及最详细的内容为止。

"分解"和"抽象"是系统开发技术中控制复杂性的两种手段。先将系统"抽象"成一个模型，此模型是有输入和输出并有系统名称的盒子，然后打开这个盒子，对它进行逐层分解，直到能被理解、可以实现为止。因此，分解的策略是自顶向下、逐层加细、由抽象到具体的过程，如图4.4所示。

2. 结构化分析的步骤

用结构化分析方法进行系统需求分析的具体步骤如下：

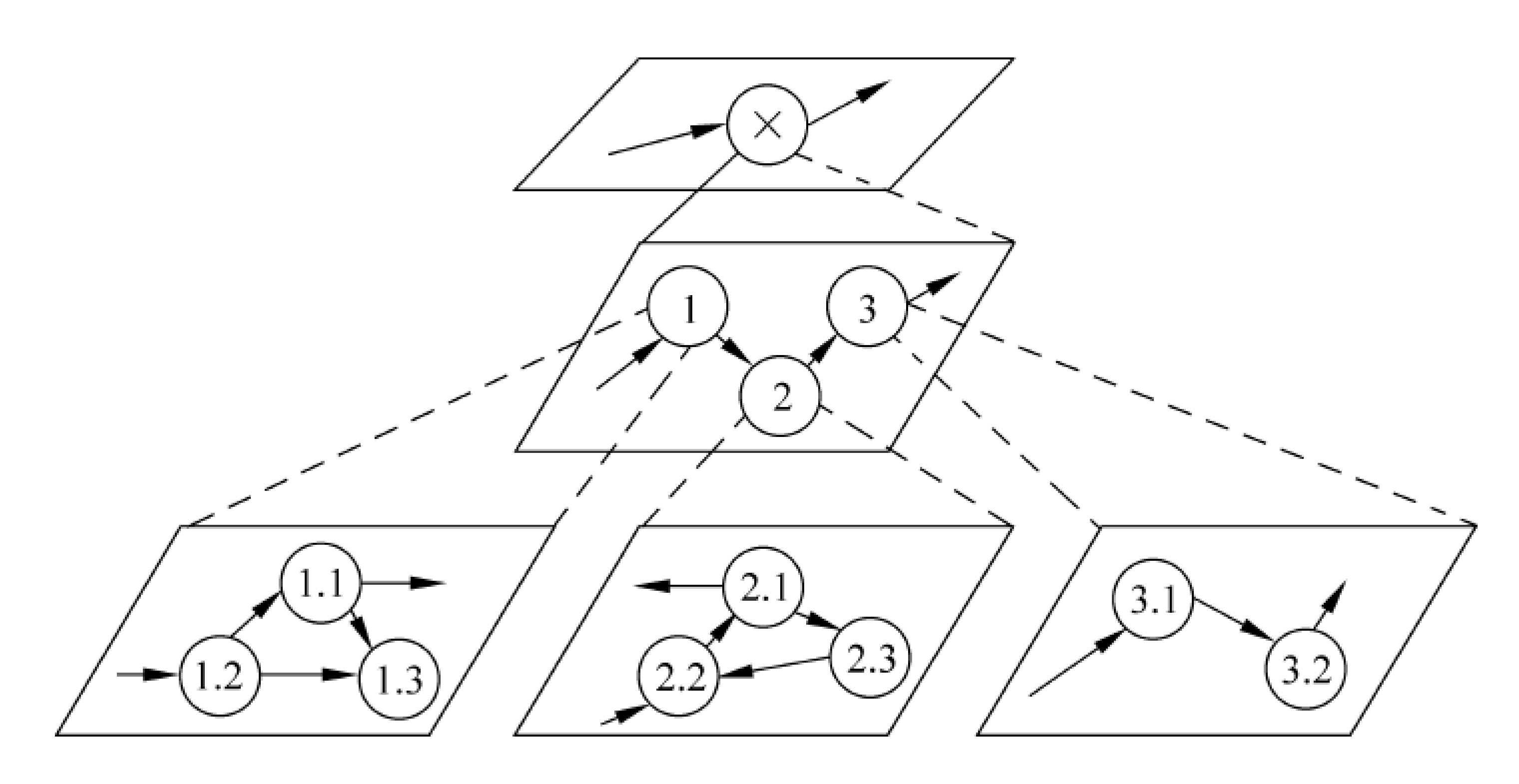


图 4.4 自顶向下逐层分解

(1) 了解当前系统的工作流程,获得当前系统的物理模型。通过对当前系统的详细调查,了解当前系统的工作过程,同时收集资料、文件、数据、报表等,将看到的、听到的、收集到的信息用图形描述出来,也就是用一个模型来反映对当前系统的理解,例如用系统流程图。

(2) 抽象出当前系统的逻辑模型。物理模型反映了系统"怎么做"的具体实现,去掉物理模型中非本质因素,抽取本质因素,构造当前系统的逻辑模型,反映了当前系统"做什么"的功能。

(3) 建立目标系统的逻辑模型。分析、比较目标系统与当前系统逻辑上的差别,明确目标系统到底要"做什么",从当前系统的逻辑模型导出目标系统的逻辑模型。

(4) 进一步补充和优化。为了对目标系统做完整的描述,还需要对逻辑模型进一步补充,包括说明目标系统的人机界面,说明至今尚未详细考虑的细节(包括出错处理、启动与结束、输入输出和系统性能方面的需求等),对系统特有的其他必须满足的性能和限制,也需要用适当的形式做出书面记录。

分析阶段结束时,系统分析员必须和用户再次认真审查系统文件,争取在系统开始设计之前,尽可能发现其中存在的错误并及时纠正,直至用户确认这个模型表达了他们的要求后,系统文件(软件需求规格说明书等)才作为用户和软件开发人员之间的"合同",而最后得到确定。

3. 结构化分析的工具

SA 方法利用图形等半形式化的描述方式表达需求,形成需求规格说明书中的主要部分。描述工具包括以下几种:

(1) 数据流图。描述系统由哪几部分组成,各部分之间有什么联系。

(2) 数据字典。定义数据流图中每一个图形元素。

(3) 描述加工逻辑的结构化语言、判定表、判定树。详细描述数据流图中不能被再分解的每一个加工。

由于分析中的主要依据是数据传递及数据变换所形成的数据流,所以结构化分析一般采用数据流图的分析方法,最终结果是产生需求规格说明书,该文档包括一套数据流图、对数据流图中的成分进行定义的数据字典及对加工逻辑的描述。

4. 画分层 DFD 图的基本原则

前面章节已对数据流图的画法和数据字典进行了较详细的说明。在此仅对画分层 DFD 图的基本原则进行说明。

画分层 DFD 图的一般原则是：先全局后局部、先整体后细节、先抽象后具体。通常将这种分层的 DFD 图分为顶层、中间层、底层。顶层图说明系统的边界，即系统的输入和输出数据流，顶层图只有一张；底层图由一些不能再分解的加工组成，这些加工都已足够简单，称为基本加工；在顶层和底层之间的是中间层，中间层的数据流图描述了某个加工的分解，而它的组成部分又要进一步分解。

画分层 DFD 图的具体原则如下：

(1) 数据守恒与数据封闭原则。这一原则是指加工的输入输出数据流是否匹配，即每一个加工既有输入数据流又有输出数据流。或者说一个加工至少有一个输入数据流、一个输出数据流。

(2) 加工分解的原则。有以下三条：

① 自然性。概念上合理、清晰。

② 均匀性。理想的分解是将一个问题分解成大小均匀的几个部分。

③ 分解度。一般每个加工每次分解最多不要超过七个子加工，应分解到基本加工为止。

(3) 子图与父图的"平衡"。父图中某个加工的输入输出数据流，应该同相应子图的输入输出相同(相对应)，分层数据流图的这种特点称为子图与父图的"平衡"。

在图 4.5 中，子图与父图不平衡。子图是父图中加工 2 的分解，加工 2 有输入数据流 R 和 M，输出数据流 T，而子图则只有一个输入数据流 N，却有两个输出数据流 T 与 S。

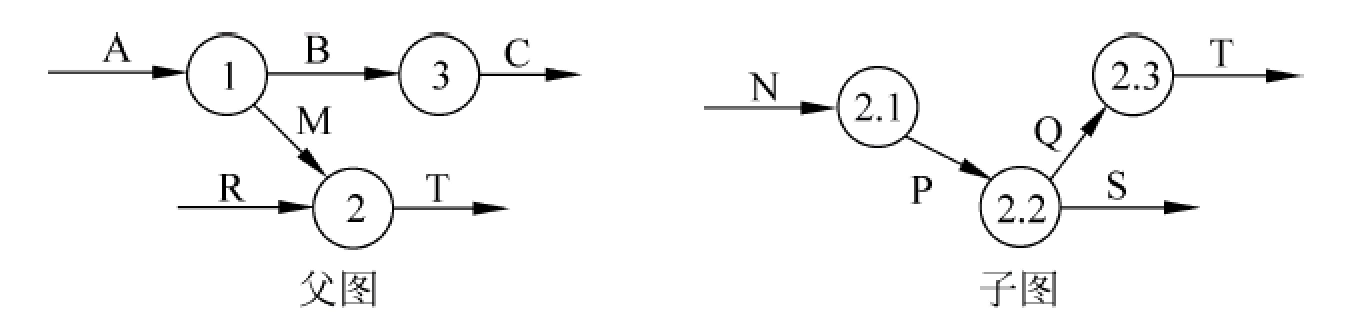


图 4.5 子图与父图不平衡

在图 4.6 中，子图是父图中加工 3 的分解，虽然表面上加工 3 只有一个输入数据流"订货单"，而子图却有三个输入数据流，但是如果"订货单"是由"客户"、"账号"、"数量"三部分组成，即有如下数据条目："订货单＝客户＋账号＋数量"，则子图与父图平衡。

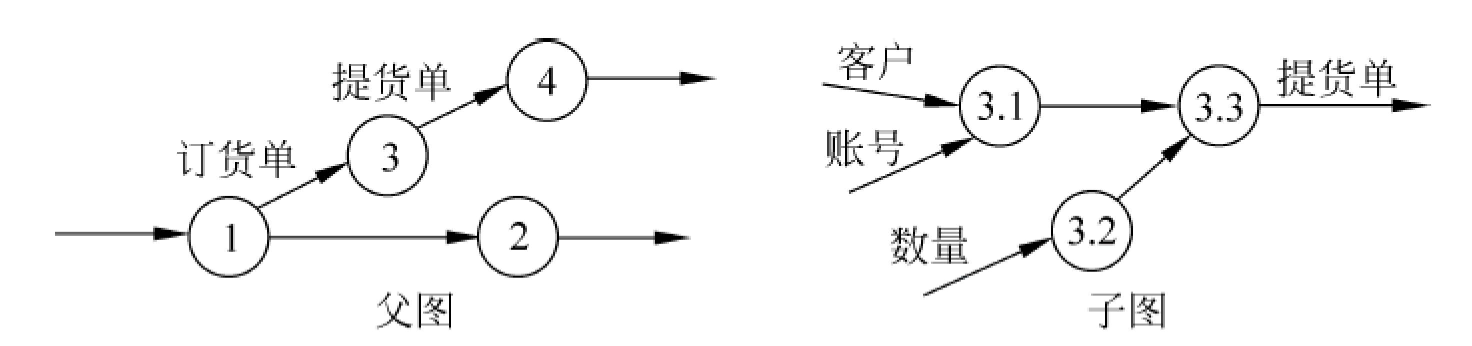


图 4.6 子图与父图平衡

(4) 合理使用文件。当文件作为加工之间的交界面时，文件必须画出来，一旦文件作为数据流图中的一个独立成分画出来，那么同其他成分之间的联系也应同时表达出来。

理解一个问题总要经过从不正确到正确,从不确切到确切的过程,需求分析的过程总是要不断反复,一次就成功的可能性很小,对复杂系统尤其如此。因此,系统分析员应随时准备对数据流图进行修改和完善,与用户达成共识,获得无二义性的需求,才能获得正确清晰的需求说明,使得设计、编程等阶段能够顺利进行。

5. 分层 DFD 图的改进

DFD 图必须经过反复修改,才能获得最终目标系统的逻辑模型(目标系统的 DFD 图)。改进的原则与画分层 DFD 图的基本原则一致,可从以下几个方面考虑 DFD 图的改进:

1) 检查数据流的正确性

- 数据守恒;
- 子图与父图平衡;
- 文件使用是否合理,特别注意输入输出文件的数据流。

2) 改进 DFD 图的易理解性

- 简化加工之间的联系(加工间的数据流越少,独立性越强,易理解性越好);
- 改进分解的均匀性;
- 适当命名(各成分名称无二义性、准确、具体)。

6. 结构化分析方法的优缺点

1) 优点

结构化分析方法是软件需求分析中公认的、有成效的、技术成熟的、使用广泛的一种方法,适合开发数据处理类型软件的需求分析。利用图形等半形式化工具表达需求简明易读,也易于使用,为下一阶段的设计、测试、评价提供了有利条件。

2) 缺点

结构化分析方法的缺点主要有以下几个方面:

(1) 传统的 SA 方法主要用于数据处理方面的问题,主要工具 DFD 体现了系统“做什么”的功能,但仅是一个静态模型,没有反映处理的顺序,即控制流程。因此,不适合描述实时控制系统。

(2) 20 世纪 60 年代末出现的数据库技术使许多大型数据处理系统中的数据都组织成数据库的形式,SA 方法使用 DFD 在分析与描述“数据要求”方面是有局限的,DFD 应与数据库技术中的实体联系图(E-R 图)结合起来。E-R 图能增加对数据存储的细节以及数据与数据之间、数据与处理过程之间关系的理解,还解决了在 DD 中包含的数据内容表示问题,这样才能较完整地描述用户对系统的需求。

(3) 对于一些频繁的人机交互软件系统,如飞机订票、银行管理等,用户最关心如何使用,输入命令、操作方式、系统响应方式、输出格式等都是用户需求的重要方面,DFD 不适合描述复杂人机界面系统的需求,SA 方法往往对这一部分用自然语言作补充。

(4) 描述软件需求的精确性有待提高。

4.4.2 面向对象方法

面向对象技术在 20 世纪 60 年代是作为一种程序设计技术提出的,到 20 世纪 80 年代

中后期，面向对象技术已经逐步发展成为软件设计和开发的主流技术，而面向对象的需求分析(Orient Object Analysis，OOA)是在20世纪90年代后期才发展起来，强调以系统中的数据或信息为主线，全面、系统、详尽地描述系统信息，建立系统信息模型，指导系统设计。

面向对象的需求分析方法能克服结构化分析方法的缺点，以对象模拟现实实体，能使用户和分析者之间很好地沟通，更容易理解需求。由于对象的封装性等特点，变化容易隔离并能较快实现。从OOA阶段到OOD(面向对象设计)阶段使用同一对象概念，开发过程中阶段的改变不需要方法的转换。最重要的是，面向对象技术中的继承机制能更好地适应重用，提高了软件开发的质量和效率。

1. 构造对象模型图

根据用户需求报告或调研报告来识别对象类、确定对象类及类之间的关联，描述对象和链的属性，利用子类对父类的继承特性构造和简化对象类，画出对象模型图，编写数据字典。对象模型图采用统一的符号表示，划分为模块，按照标准格式形成计算机文档，供开发人员查阅。

对象包括物理实体和概念，一般以名词形式出现，但并不是所有名词都是对象，但问题描述的中心实体具有成为对象的最大可能性，因为中心实体具有独立的特征和功能。半自动识别对象的方法有两种：一种是用计算机程序扫描用户需求报告，摘录出所有名词，由分析人员确定可作为对象的名词；另一种是开发中采用的频度法，用程序扫描，找出所有名词，计算这些名词在用户需求报告中各自出现的次数，展示给分析员参考产生对象名词。

关联表示了对象类之间的联接关系，常用描述性动词或动词词组表达。识别方法可根据已经识别出的对象，从用户需求报告中查找这些对象之间的语法联接关系来确定。

数据字典是内容广泛的技术文件，包括被开发系统所有对象实体的精确含义、抽象类包含的对象实体的范围、类的属性和操作描述、关联和联接的定义及说明等。所有无法用其他方式描述清楚的问题，都可以在数据字典中予以详细说明。编制数据字典的材料来源是用户需求报告、开发人员和用户的交互信息及其他相关资料。字典由词条名、描述和关联词组成。描述是词条详细说明；关联词是指与本词条相关的其他词汇。

对象模型的符号表示如图4.7所示。

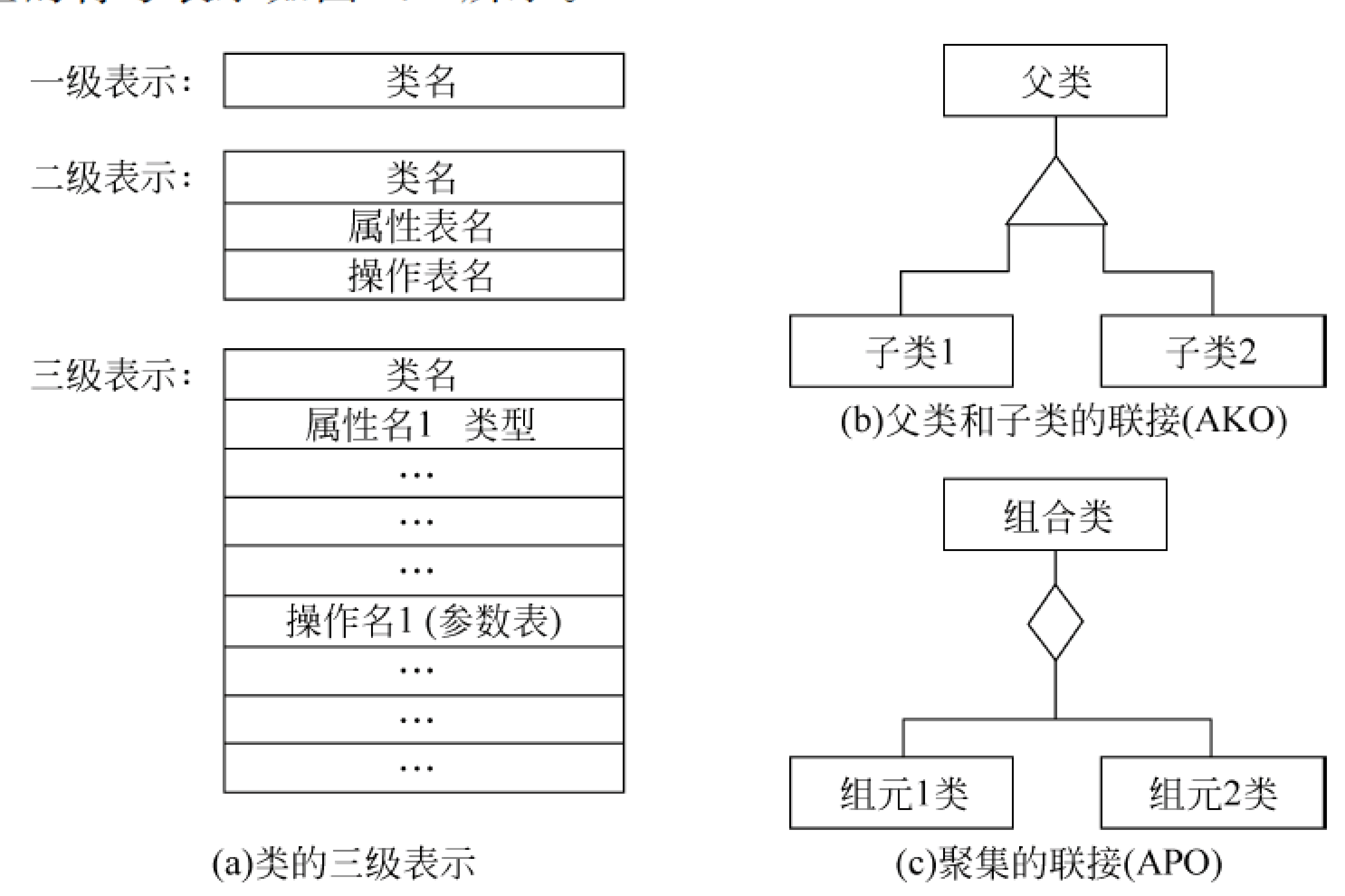


图4.7 对象模型的符号表示

图 4.7(a)中类的三级表示主要用于对象类的不同抽象级。类的命名要概括类的本质特征。其中：

属性表名＝类名-属性表

属性表＝属性名＋类型＋描述信息

操作表名＝类名-操作表

操作表＝操作名＋参数表＋返回类型＋描述信息

图 4.7(b)表示父类和子类的联接方法，图 4.7(c)表示组合类和组元类的联接方法。

父类和子类、组合类和组元类都采用自上而下分解联接。等腰三角形的顶角通过直线段和父类框的底部联接，而底边通过直线段和子类的顶部相联。菱形的上角通过直线段和组合类的底部相联，而下角通过直线段和组元类的顶部相联。这样就可从图示中直接区分父类和子类、组合类和组元类。

所有类之间的关联联接点应在类框的左右边线上，以便整个类对象图更加清晰。关联的符号表示如图 4.8 所示。

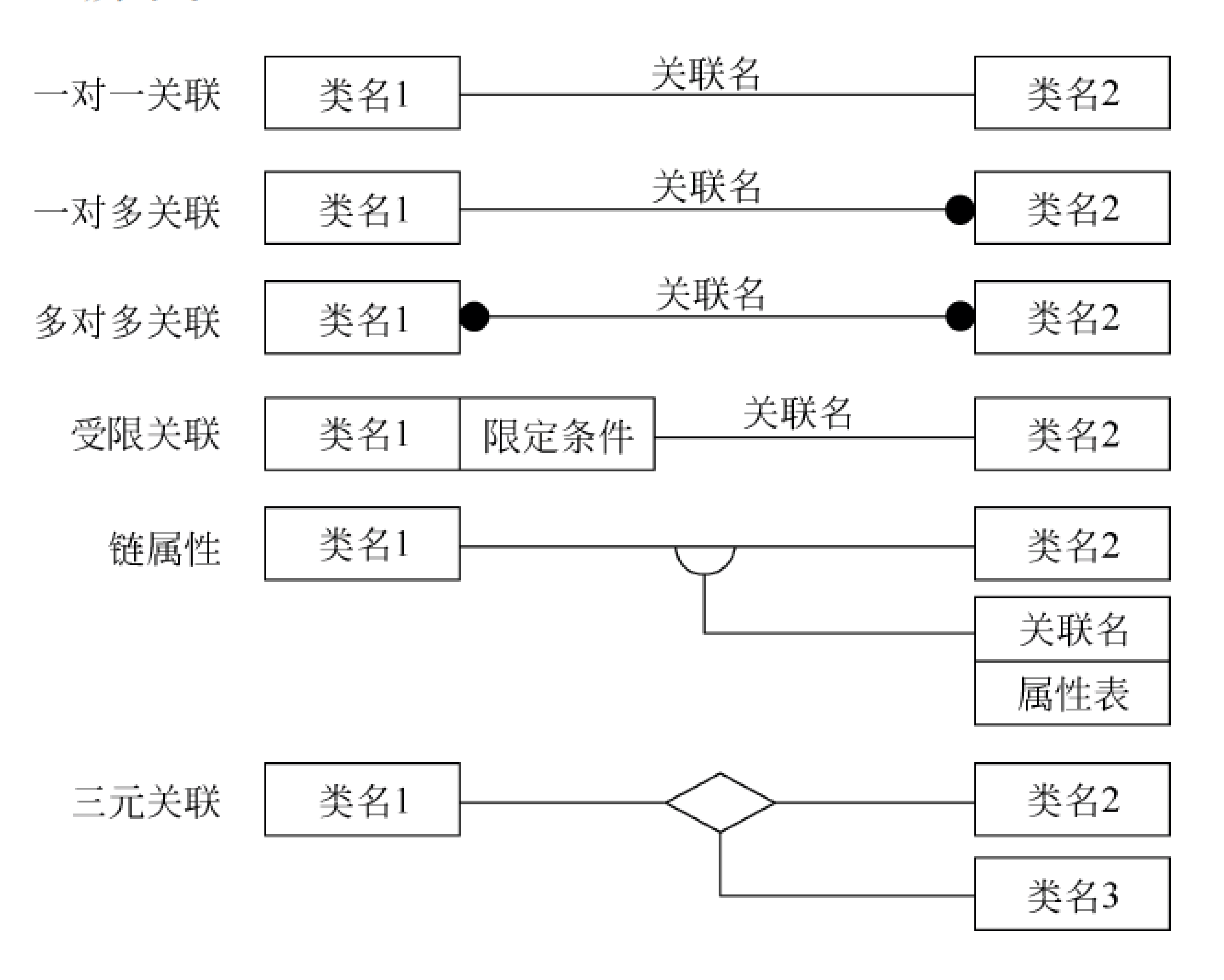


图 4.8 关联的符号表示

2. 创建动态模型

动态模型描述系统中与时间和操作次序有关的属性，主要包括触发事件、事件序列、事件状态、事件状态的组织、表现应用的控制逻辑等。一般采用状态图描述系统的状态模型。一张状态图描述一个类中所有对象的状态和事件的正确顺序。创建动态模型需要经过撰写脚本、勾画用户接口模式图和事件跟踪图、描述对象状态图等复杂的逐步逼近的过程，以确保整个交互行为流的清晰和正确。资料来源是用户需求报告及相关技术资料。

创建动态模型各个过程的工作内容如下：

(1) 撰写用户和系统之间交互活动的脚本。脚本是一个事件序列，每当系统中的对象与用户发生信息交换时就产生一个事件，所交换的信息就是该事件的参数。对于每一个事件，应该确定触发该事件的动作对象、外部显示形式和该事件的参数。脚本的描述方法为：以对象为中心，以时间为顺序，以事件为线索。脚本的格式和一般说明性文件的格式相同。

(2) 依据交互脚本，确定用户接口模式图，即界面显示的大体布局，以不丢失交互事件为原则，用户界面的美化暂不考虑。用户接口模式图可用带事件名的矩形表示。

(3) 把脚本描述图示转化为对象间的事件流图。事件流图的符号表示如图 4.9 所示。

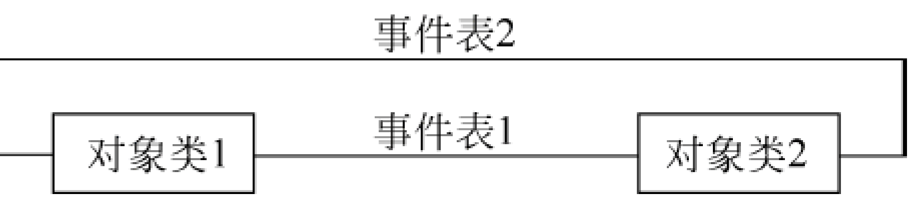
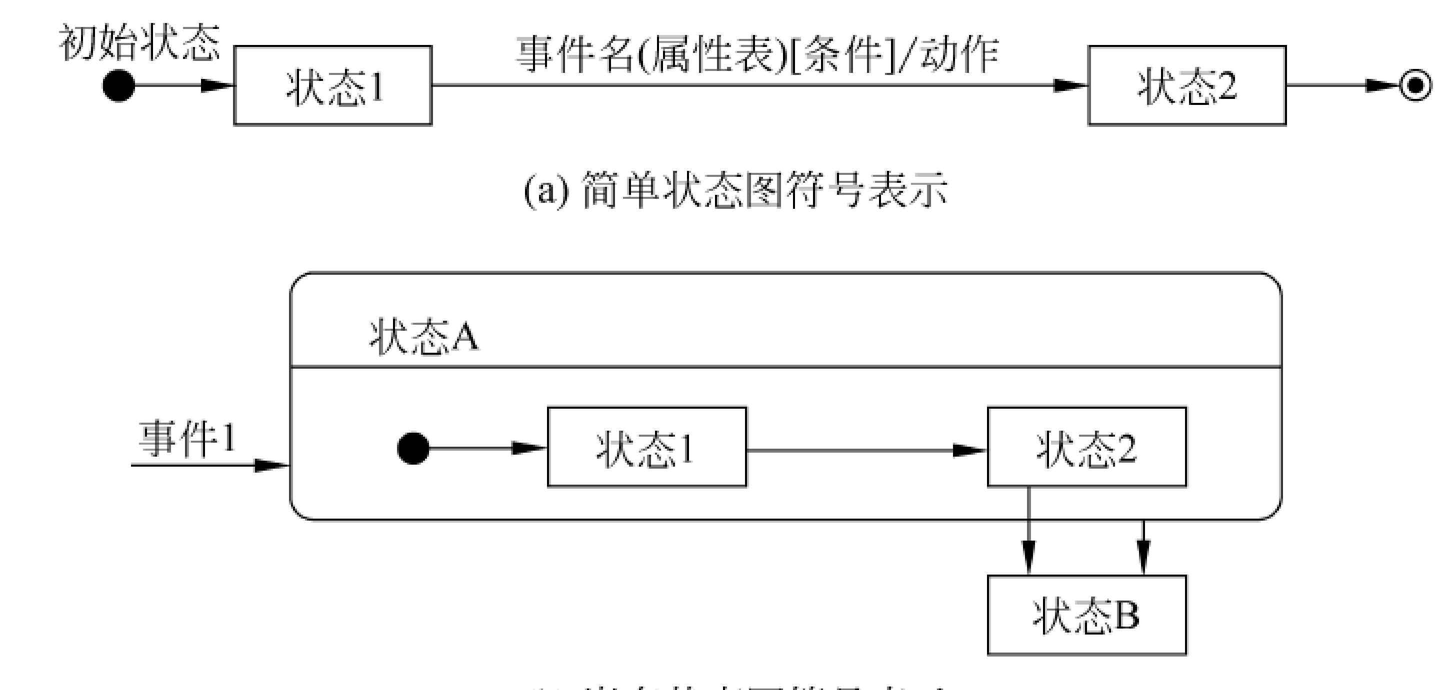


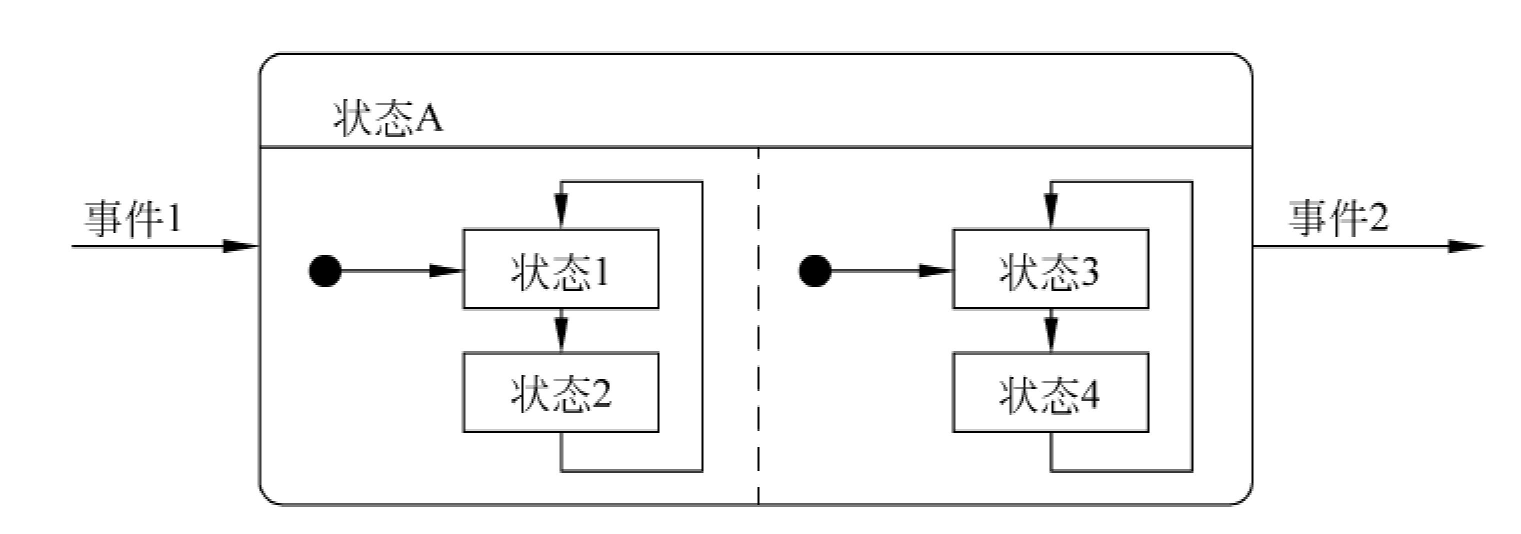
图 4.9　事件流图的符号表示

事件采用事件序号、事件名、发送对象类、接收对象类四个字段的记录描述。根据对象间的事件流图，以对象为单位，建立状态图。状态图符号表示如图 4.10 所示。

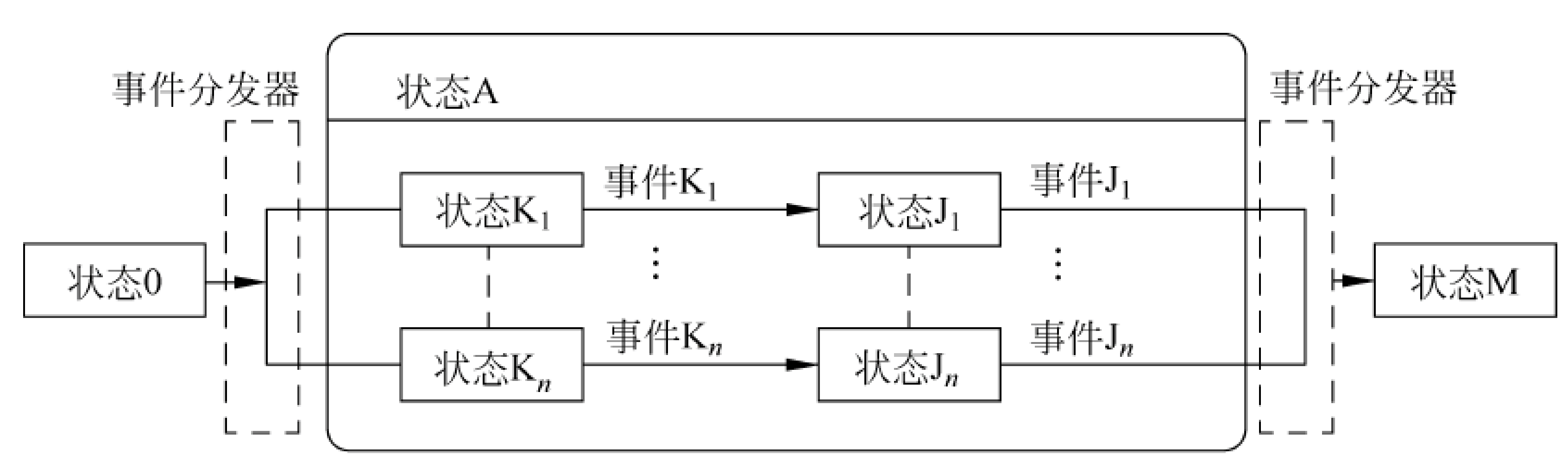


(a) 简单状态图符号表示

(b) 嵌套状态图符号表示



(c) 并发状态图符号表示



(d) 分离与同步控制状态图符号表示

图 4.10　状态图符号表示

圆角矩形框是状态的表示符号，带箭头的线段是转换的表示符号。

实心圆点表示初始状态，应当在实心圆点上作标注，以说明不同的初始条件。同心圆点表示最终状态，同样要说明终止事件。事件后的说明可以是任选的，其中的圆括号、方括号、左斜杠分别是属性参数表、条件和动作的说明符号，标注时不要遗漏。

3. 构造功能模型

功能模型描述对象模型的操作和约束,解释动态模型的动作。功能模型由数据流图(DFD)组成。DFD的处理用对象类上的操作实现。数据流将一个对象的处理或输出同另一个对象的处理或输入联系起来。动作对象和数据存储都是对象,和对象模型中的概念相同。

构造功能模型的依据是用户需求说明、对象模型和动态模型及其他技术资料,在必要时,还要深入用户进行更细致的调查。

构造功能模型时,需要找出应用系统与外部世界之间事件的参数,即输入输出值,建立数据流图来说明从输入值到输出值的变化过程,把处理功能、各对象间的约束条件和优化标准等的描述补充到数据字典中。

在构造功能模型的过程中,要尽量从用户需求说明中寻找数据产生的实体及各实体间的数据流向,处理和处理之间、处理和实体之间的数据流向,从而体现出从不同侧面观察应用域,便于发现问题。功能模型符号如图4.11所示。

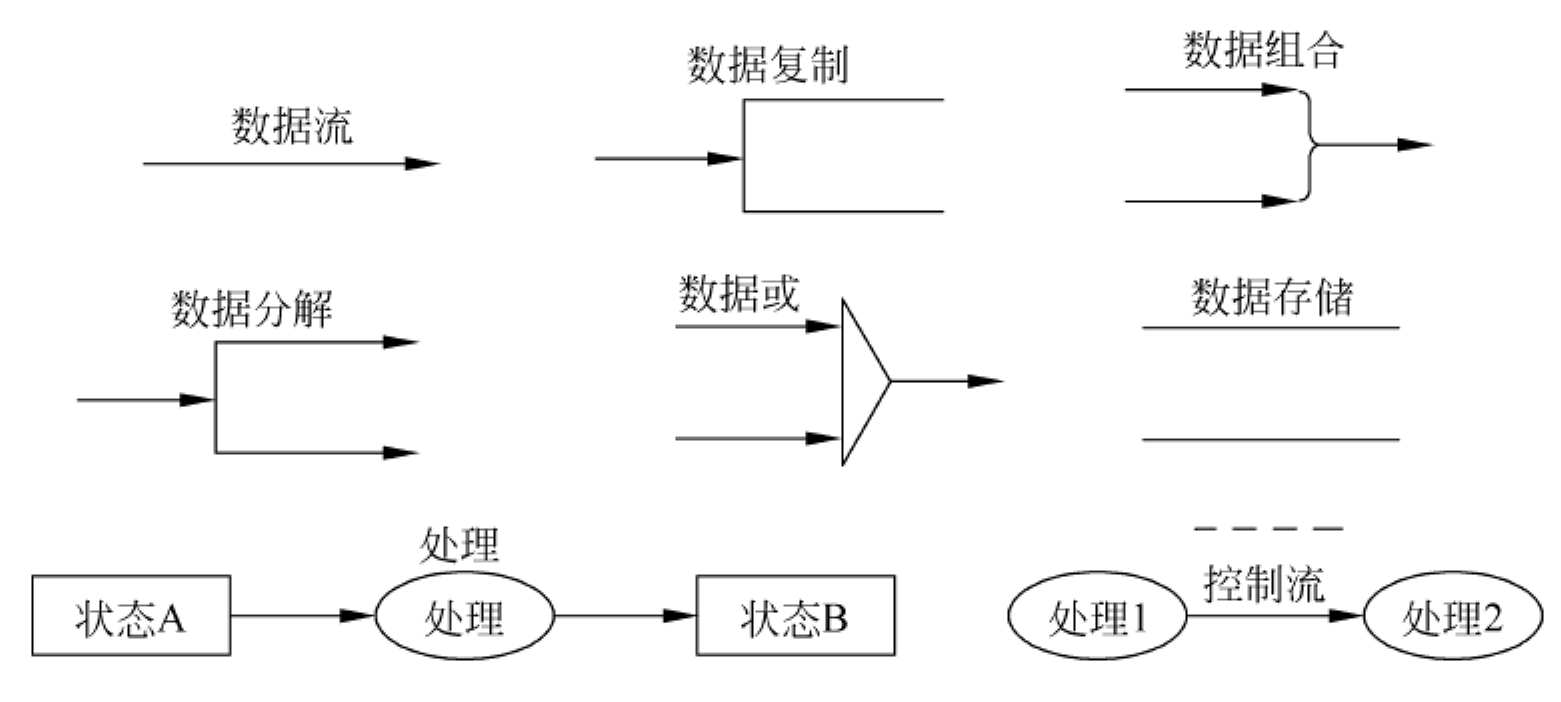


图4.11 功能模型符号表示

4. 验证评审

构造三个模型不是简单的一次性过程,需要反复进行验证、细化和完善。验证内容包括以下几个方面。

(1) 正确性:指类描述、对象定义及对象之间关系的说明是否正确,各对象的状态变化流程是否正确。

(2) 完整性:指模型中是否出现未定义的对象,并对照问题域检查是否遗漏了某些对象,各对象是否完整,该做的事是否都做了,不该做的事都要删去。

(3) 一致性:指同一类对象是否有不同定义,各模型之间同一事务、同一活动的描述是否一致。

经过开发人员反复检查验证后,组织召开评审会议。评审步骤是:

(1) 软件开发人员以"用户接口模式图"为线索,向应用领域专家详细介绍"系统能干什么",给出效果显示,并认真听取意见。

(2) 以对象模型图为主导,结合动态模型和功能模型,向软件开发领域专家详细介绍对象模型的产生过程,了解专家对分析模型的正确性与合理性方面的见解。

(3) 根据评审结果,对相关文档做进一步的修改完善。

(4) 最后形成完整的需求分析文档。需求分析文档包括用户需求报告、对象模型图、动态模型图、用户接口模式图、全局事件流图、数据字典等。

4.4.3 原型方法

当用户和开发人员一起确定需求时，如果用户不能确定自己的真正需求，开发人员也不能确定用户的需求，这时可采用原型方法进行需求分析。按照用户需要，快速形成一个操作流程界面，可以只是一个框架，具体功能没有完全实现，以便快速与用户就需求达成一致。原型分析法主要考虑系统的功能需求，很少考虑非功能需求。有关原型模型的基本过程、软件支撑环境、优缺点等已在前面章节中说明，本小节讲述使用原型的目的、原型建立技术、原型分类等问题。

1. 使用原型的目的

软件原型通常仅仅是真实系统的一部分或一个模型，是对新产品的部分实现。使用原型的目的如下：

(1) 明确并完善需求。原型作为一种需求工具，用户通过对原型的评价可以指出需求中的许多问题，在真正开发产品之前，可以用较低的费用来解决这些问题。

(2) 探索设计选择方案。原型作为一种设计工具，可以探索不同的用户界面技术，使系统达到最佳的可用性，并且评价可能的技术方案。

(3) 发展为最终的产品模型。原型作为一种构造工具，是产品最初子集的完整功能实现，通过一系列小规模的开发循环，可以完成整个产品开发。

(4) 需求分析阶段建立原型的主要目的是解决在产品开发早期阶段的不确定性因素，通过建立原型，有助于说明和纠正这些不确定性因素。

2. 原型建立技术

原型建立有以下一些技术：

(1) 可执行规格说明。这是基于需求规格说明的一种自动化技术，使用这种技术可以直接观察用语言规定的任何系统的功能和行为。

(2) 基于脚本的设计。脚本是用户界面的原型，用来模拟在系统运行期间用户经历的事件，提供了“输入—处理—输出”的屏幕格式和有关对话模型。软件开发者能够给用户显示系统逼真的视图，使用户得以判断是否符合实际应用。

(3) 自动程序设计。在程序自动生成环境的支持下，利用计算机实现软件开发。可以自动或半自动地把用户的非过程性问题、规格说明等转换为某种高级语言程序。

(4) 专用语言。专用语言是应用领域的模型化语言。在原型开发中使用专用语言可方便用户和软件开发者对系统特性进行交流。

(5) 可复用的软件。利用可复用的模块，通过适当组合，构造原型系统。为了快速构造原型，这些模块必须有简单而清晰的界面，尽量不依赖其他模块或数据结构，并具有一些通用的功能。

(6) 简化假设。简化假设使设计者迅速得到一个简化的系统。尽管这些假设可能实际上并不能成立，但可以使开发者的注意力集中在一些主要方面。修改一个文件时，可以假设这个文件确实存在。存取文件时，待存取的记录总是存在。一旦计划中的系统满足用户所

有要求，就可以撤销这些假设，并追加一些细节。

3. 原型分类

原型可以有以下两种分类方法：

1）水平原型和垂直原型

（1）水平原型也叫行为原型，可以探索预期系统的一些特定行为，并达到细化需求的目的。当用户考虑原型中提出的功能是否能完成各自的业务时，水平原型使用户探讨的问题更加具体化。

（2）垂直原型也叫结构化原型，实现一部分应用功能。当不能确信构造软件的方法是否完善或者需要优化算法、评价一个数据库的图表或测试临界时间需求时，就要开发一个垂直原型。垂直原型更常用于软件设计阶段以减少风险。

2）抛弃原型和进化原型

构造原型之前，需要与用户充分交流，并做出明确的判断。评价原型以后，就要决定是抛弃掉原型还是把原型进化为最终产品的一部分。

（1）抛弃型原型。抛弃型原型要忽略很多具体的软件构造技术，而强调在健壮性、可靠性、可维护性和性能等原则下迅速实现软件。基于这个原因，不能将抛弃型原型中的代码移植到产品系统中，除非达到产品质量代码标准。如果遇到软件需求中的不确定性、二义性、不完整性或含糊性时，最合适的办法是建立抛弃型原型，需要解决这些问题以减少在继续开发时存在的风险。原型可以帮助用户和开发者思考如何实现需求，并可以发现需求中的漏洞，还可以使用户判断出这些需求是否可以完成必要的业务过程。

（2）进化型原型。与抛弃型原型相对应的是进化型原型。在已经清楚地定义了需求的情况下，进化型原型为开发渐增式产品提供了坚实的构造基础。进化型原型是螺旋式软件开发生命周期模型的一部分。与抛弃型原型的快速、粗略等特点相比，进化型原型一开始就必须具有健壮性和产品质量级的代码。因此，对于描述相同的功能，建立进化型原型比抛弃型原型所花的时间要多。

4. 综合运用多种原型方法

在软件开发过程中可以综合使用多种原型方法，如图 4.12 所示。例如，可以利用从一系列抛弃型原型中获得的知识来精化需求，然后通过一个进化型原型系列可以渐增式地实现需求。

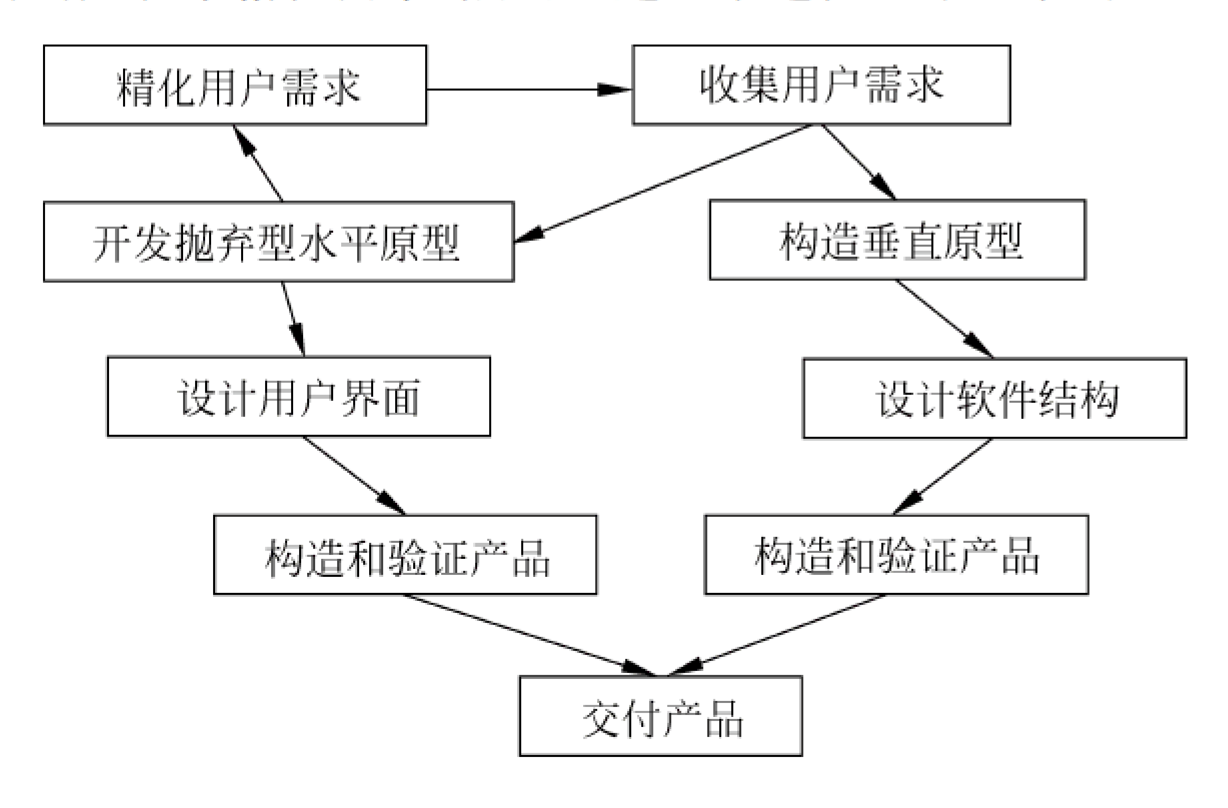


图 4.12　综合使用多种原型方法

4.4.4 用例建模

用例建模是面向对象软件设计和开发方法中的一项技术。Ivar Jacobson 于 20 世纪 80 年代中期提出用例的概念，经过多年发展，已经成为面向对象软件开发技术中的一个重要部分，被有效地应用到软件开发的需求分析中，并且对整个软件项目的开发进程都具有明显的指导和推动作用。

1. 使用用例表示软件需求

1）软件需求

在需求分析过程中，需求一般可以分为两类：功能需求和非功能需求。功能需求是系统必须实现的行为，不需要在条件中加入物理约束，指定了系统的输入和输出行为；非功能需求指定了系统必须具有的其他性质，例如可用性、可靠性、整体或局部性能、可支持性等，描述系统特征或系统的环境特征。

2）用例

用例是参与者与系统的交互。用例代表了系统为参与者执行的有价值操作，用于表达系统的功能需求和行为，获取功能需求。用例可以捕获在引发系统某些行为时用户所做的操作，同时还可以捕获系统在提供所需行为时所做的操作。用例清晰地定义了系统在满足用户目的方面具有的责任，以及用户是如何支持系统的。用例的图形表示如图 4.13 所示。

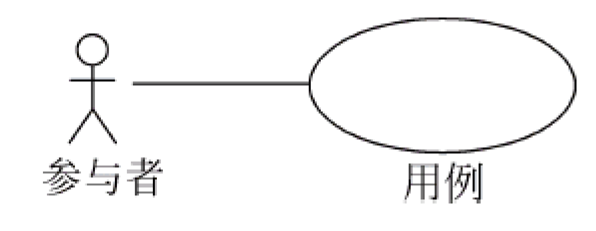


图 4.13 用例的图形表示

3）使用用例表示软件需求的原因

常规的需求描述方法侧重声明需求，陈述系统应该怎么样、应该为用户做什么。而用例体现参与者需要系统完成何种操作，强调实现用户的真正目标，以一种简单而有效的方式表达系统行为，更体现系统对用户的价值；用例把需求书写者的注意力从系统功能列表转到用户操作上，因为用户更关心系统能提供何种有价值的操作，这是使用软件的目标。

用例将软件需求放在应用环境中，将功能需求放入实际操作的用户环境中，当系统与用户或其他系统交互时，可以描述系统行为。对非功能需求使用常规的声明需求描述是比较合适的，并且可以附加到用例中。

2. 用例模型

用例模型是所有用于描述指定系统的用例、参与者、用例-参与者关联关系的组合。在 UML 中用例模型的定义是：用例模型可用于描述系统的功能性需求或者用例的其他分类。在软件开发过程中，使用模型对现实进行简化、概括，从而加强对真实构建系统的理解。

用例模型的基本构件包括以下内容。

（1）用例名称：所有用例名称应该表明它与参与者的交互结果，为了便于理解，可以使用多个单词，并且是唯一的。

（2）简要描述：对用例的角色、目的的简要描述。

（3）事件流：从用例角度对系统行为进行文本描述，开发人员应该能够理解这些描述，一般由基本流、备选流和子流构成，有时也包括可视化的事件流。

(4) 前置条件：定义用例启动的系统约束，以文本形式描述。

(5) 后置条件：定义用例终止后的系统约束，以文本形式描述。

(6) 扩展点：用例事件流中的一个位置列表，在这些位置上可以插入附加行为。

(7) 关系：用例参与的一些关系，比如通信关系。

(8) 示意图：演示用例的各个方面，比如事件流的结构或用例涉及的关系。

(9) 特殊需求：收集用例中不在事件流内考虑的，但在设计和实现过程中都要考虑的需求(例如非功能性需求)。

用例模型的这些构件既有图形表示，又包含文字陈述，因此可以更直观地表达用户需求。在用例中最重要的属性就是事件流，它描述了系统和参与者如何协作传递用例体现的需求价值。

3. 实例分析

(1) 问题描述。汽车维修业务流程的工作流描述是：前台接待人员根据车辆进厂后检测情况，确定车主基本资料以及车辆维修内容，填写派工单，交给维修人员；维修人员根据派工单，向仓库管理员提供出库的配件和材料清单；仓库管理员根据库存情况决定配件和材料出库，同时还要根据需要处理入库信息；维修完毕后，检验员对维修车辆进行质量检验，合格后通知财务人员结算维修费用，车辆出厂。

(2) 用例建模。根据问题描述，可以将系统分别以下列几个用例描述系统需求：派工、配件的出库与入库、出厂检验和费用结算。需求用例模型如图 4.14 所示。

总体的用例模型对需求描述不够详细，有必要进一步细化，为每个用例分别建模，更准确地把握系统需求。以派工为例进行细化，用例图如图 4.15 所示。

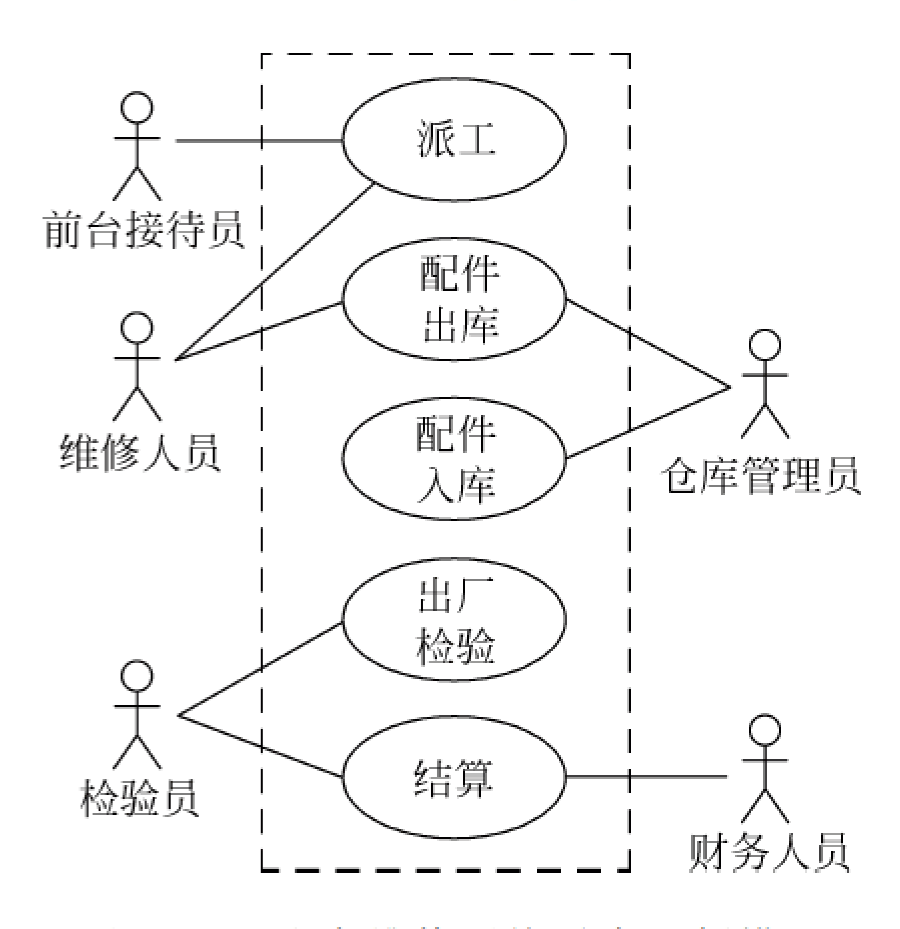


图 4.14　汽车维修系统需求用例模型

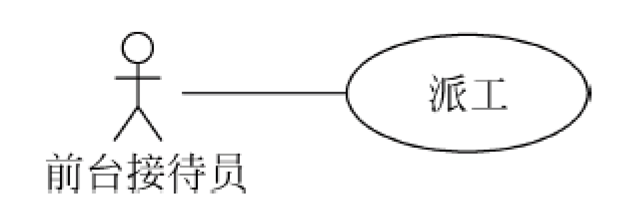


图 4.15　派工用例

(3) 用例简述。根据待修车辆进厂的检测结果，开具车辆维修派工单。

(4) 前置条件。待修车辆经过专业技师进行故障检测；应用系统与数据库服务器保持有效连接；系统初始化数据准备完毕；参与者具备操作的系统权限。

(5) 基本流。

① 前台接待员根据汽车进厂后的故障检测结果，填写派工单，开始启动用例。

{显示备选的车辆基本信息}

② 系统显示要求填写的汽车基本信息，包括车牌号、车型、发动机号等。

{显示已有的客户编号}

③ 如果是原有客户则直接选择，否则填写客户基本信息，包括客户编号、名称、电话、住址等。

④ 系统显示要求填写的派工单内容，包括派工单号、送修类型、派工/完工日期时间、维修人、发票编号等。

{显示系统维修内容信息}

⑤ 选择维修内容，包括维修类别、维修项目等。

⑥ 系统验证输入数据，确认派工单后，保存所有信息。

⑦ 用例终止。

备选流：a. 新客户信息处理，如果客户不是系统原有客户，则系统要将新的客户信息加入到客户信息表中；b. 如果保存信息操作失败，将详细出错信息返回显示，提示操作员，用例结束；c. 如果维修内容填写操作错误，与实际不符，可以更改维修信息，继续用例执行。

特殊需求：如果派工单内容保存失败，则其他信息包括汽车基本信息、客户基本信息、维修内容信息将放弃存储，保证数据库的完整性和一致性。

在用例的基本事件流描述中，大括号（{ }）部分表示事件流的扩展点，用以插入附加的行为。在派工的用例模型中考虑了用例的需求环境，因此问题描述包含的内容有所增加。

4. 总结

用例建模得到的功能需求明确规定了参与者执行的特定任务，使用户能清晰看到系统提供的有价值操作。使用用例复杂度帮助人们理解复杂问题。

需要说明的是，用例只是一种需求文档，不涉及系统设计、界面设计特性列表以及测试。用例主要捕捉系统功能性需求，而对非功能性需求，需要其他的需求模型获得，或者使用补充规范加以描述。使用用例建模时，要理解涉众和用户群体，确定系统的参与者和系统边界；要理解参与者并不一定总是人；如果系统边界定义不清，就会造成系统功能定义的含糊不清。

4.5 需求分析变更

由于软件项目自身的特点，需求分析过程中发生的变化较多。要有效地减少需求变更，要求需求人员和用户方都要有正确的思想认识，在开发过程中积极合作，通过协议制定各方要承担的责任和履行的义务，树立减少需求变更的意识，并积极采用新技术支持来应对变更。需求分析文档要规范，合作双方要精诚合作，不断分析和总结实践经验，努力提高技术水平，确保软件开发工作按照正确的方向进行，最终获得成功。本节首先分析需求变更的原因，然后指出相应的对策。

4.5.1 需求变更的原因

需求变更的原因主要有以下几点：

(1) 问题域的复杂性越来越高。我国正处在新的历史时期，国家大力提倡管理创新和科技创新，企业的各种标准、流程、事务都处于重组规范的过程中，因而系统需求不明确、处于不断变化的状态，导致问题域的复杂性越来越高。

(2) 交流障碍。需求分析过程涉及人员具备的背景知识不同。需求人员一般是以软件及其相关专业为主，而用户方是以自身业务为主，考虑问题的角度不同，扮演的角色不同，认识上容易混淆，产生误解，相互之间的交流存在一定困难。

(3) 完整性问题。因专业背景、技术等方面的原因，导致用户很难准确地把系统需求表达给需求分析人员，业务方面的局限性导致开发方也很难准确获取用户方的真实需求。这样的结果就导致用户对问题的陈述不完备，需求也就不完整。

(4) 变更的需求引起更多的变更。用户方对需求的重视程度不够，对自身业务的抽象程度不够，导致需求不够细化明确，经常变更。变更的需求又导致更多的变更，引起连锁反应，最终陷入死循环，导致后续工作不能按计划完成，影响整个系统开发。

4.5.2 相应对策

软件需求变更是不可避免的，应该以正确的心态去面对变更，解决变更带来的一系列问题。在进行软件需求分析的过程中，可以从以下几个方面来积极地应对需求变更：

(1) 思想认识方面。好的软件产品取决于好的需求分析，要培养正确的需求意识，用户和开发人员之间要有良好的交流合作。同时，开发人员更应该发挥积极主动作用。

① 需求分析人员的思想意识。需求分析人员的思想意识主要包括以下几种：应对需求变化的意识，随着用户对系统需求认识的不断加深和业务流程的不断变化，需求也在不断地增加和变化，这就要求需求人员正确对待用户需求的变化，运用新技术主动应对需求的不断变化；寻求用户支持的意识，因为不可能一次就完全了解用户需求，而且在系统开发过程中还需要用户不断地参与，必须寻求用户支持；维权意识，在系统开发过程中，为了避免出现和减少与用户间的摩擦，合作双方应该签订协议，明确规定双方的责任和义务，在一定程度上减少双方之间的摩擦和随意的需求变更。

② 用户的思想意识。用户应意识到质量差的需求可能导致严重后果，因此要充分认识需求分析的重要性，主动参与到需求分析中去；同时，用户也要意识到由于技术、人力等资源的限制，计算机并不能解决当前存在的所有问题，并且软件产品开发需要一定时间，向开发人员提出系统需求时，不能毫无边界；最后，用户方还要有减少需求变更的意识，因为需求变更会增加开发成本并延长工期。

(2) 项目管理方面。做好需求调研前期的准备工作。在需求调研前将掌握的资料进行汇总，制定需求分析计划，对项目参与人员进行培训。

① 划定项目范围，拟定协议。明确项目的范围可以防止在开发过程中范围不受控制地扩展，同时，要与用户方拟定协议，明确指出项目的范围、特性和功能界限。

② 需求版本的控制。应该由专人负责需求文档的更新、管理、发布和版本控制等，组内

每个成员能够得到当前版本，每个发布的需求文档要保留修正版本的历史状况，包括变更内容、日期和原因等。

(3) 技术支持方面。当需求变更时，为了更好地适应变化，需求人员应采用技术对策，比如原型法、敏捷开发方法和软件复用技术等，还可以采用需求管理工具，常用的有CA公司的CA-Super Project & Project Software和Rational公司的Analyst Studio等。

4.6 需求分析验证

需求分析验证是指在需求分析阶段后期，通过一定的途径和手段，对初步确定的软件需求的正确性和可行性进行验证，确定正确和可行的软件需求，排除含糊、不实际和不可行的软件需求。

需要明确，并不是所有需求都可以验证，也并不是所有需求都需要验证。实际上有很大一部分需求事先无法验证，或者事先无需验证。需求验证最彻底、最有效的方法是实际开发出来的软件系统，因为真正的软件系统才是对需求的完整实现。需求分析阶段的需求验证是指对一些重要的、把握不准的需求进行验证，减少软件开发风险，提高开发的成功率。

4.6.1 需求分析验证的方法

需求分析验证的方法很多，在此仅介绍常用的几种。

(1) 自查法：由需求分析人员对自己完成的软件需求进行审核和验证，纠正需求中存在的问题。自查法又可以分为多种具体方法。

① 小组审查法：由一名分析人员向开发小组中其他人员介绍软件需求，小组中的成员进行提问，由介绍人员进行解答。在介绍过程中，可能会发现并澄清许多潜在的需求问题。实践证明这是一种十分有效的方法。

② 参照法：对系统中存在的有些可疑性需求，在系统内部无法验证其可行性时，可以参考其他系统，如果发现在其他系统中有相同或相似的需求，并且已经实现，那么就可以证明这种需求是可行的。

③ 逻辑分析法：由分析人员按照需求与业务、需求与目标、需求相互之间的逻辑关系进行逻辑论证，找出在逻辑上存在矛盾或不一致的需求进行重点分析。

(2) 用户审查法：用户是需求的提出者，也是软件系统的最终使用者，因此，由用户来审查需求是最有权威的。分析人员把需求分析文档提交给用户，有条件时可以同时编写一份针对需求的用户使用说明一并提交给用户。用户通过对需求文档的阅读，找出不符合用户意图或用户认为不能实现的需求，双方再对这些有争议的需求进行讨论，最后达成一致意见。

(3) 专家审查法：聘请业务领域、软件领域、政策、法律等方面的专家，对软件系统需求进行审查。专家能够对用户或分析人员存在争议的需求以及隐藏着重大问题的需求进行识别和判断。

(4) 原型法：对存在争议或拿不准的需求，通过建立原型进行验证，以确定需求的正确性。原型法是验证需求的一种十分有效的方法，同时也是帮助用户理解需求的一种好方法，

但要求有原型生成环境的支持。

4.6.2 需求分析验证的内容

需求分析验证的内容包括以下几个方面：

(1) 需求的正确性。开发人员和用户都进行验证，以确保将用户的需求充分、正确地表达出来。每一项需求都必须准确陈述欲开发软件的功能。做出正确判断的参考是需求的来源，如用户或高层的系统需求规格说明书。若软件需求与对应的系统需求相抵触，则是不正确的。只有用户才能确定用户需求的正确性，这也是一定要有用户参与的原因。

(2) 需求的一致性。需求的一致性是指需求分析的最终结果与其他软件需求或高层(系统、业务)需求不相矛盾。在开发前必须解决所有需求的不一致性部分，验证任何冲突和含糊的需求，使之没有二义性。

(3) 需求的完整性。验证是否所有可能的状态、状态变化、转入、产品和约束等都在需求分析文档中描述，不能遗漏任何必要的需求信息。注重用户任务而不是系统功能，将有助于避免不完整性。

(4) 需求的可行性。每项需求都必须在已知系统和环境限制条件下是可以实现的。为避免不可行的需求，最好在获取需求过程中始终有一位软件工程小组的成员与需求分析人员一起工作，由他来检查技术可行性。

(5) 需求的必要性。验证每项需求都是用户需要的，每项需求都应把用户真正需要的内容和最终系统需遵循的标准记录下来。必要性也可以理解为每项需求都是编写文档的“根源”，每项需求都能回溯到用户的输入。

(6) 需求的可检验性。验证是否能写出测试用例来满足需求，检查每项需求是否能通过设计测试用例或其他验证方法来确定产品按需求实现了。前后矛盾、不可行或有二义性的需求是不可验证的。

(7) 需求的可跟踪性。验证需求是否是可跟踪的，应能在每项软件需求与其根源和设计元素、源代码、测试用例之间建立联接，这种可跟踪性要求每项需求以一种结构化的、细粒度的方式编写并单独标明，而不是大段落地描述。

4.7 需求管理

4.7.1 需求开发与需求管理的界限

软件开发中遇到的许多问题，都是由于收集、编写、协商、修改产品需求等过程的不正确做法带来的，例如非正式信息的收集、未确定或不明确的功能、未发现或未经交流的假设、不完善的需求文档、突发的需求变更过程等。所以说，完成需求开发并形成规格说明仅是需求成功的一半，开发人员必须能够真正把用户的所有需求应用到产品中，并能够有效地控制需求变更，才能保证需求与设计的一致性，准确实现既定的需求。需求开发与需求管理的界限可用图 4.16 表示。

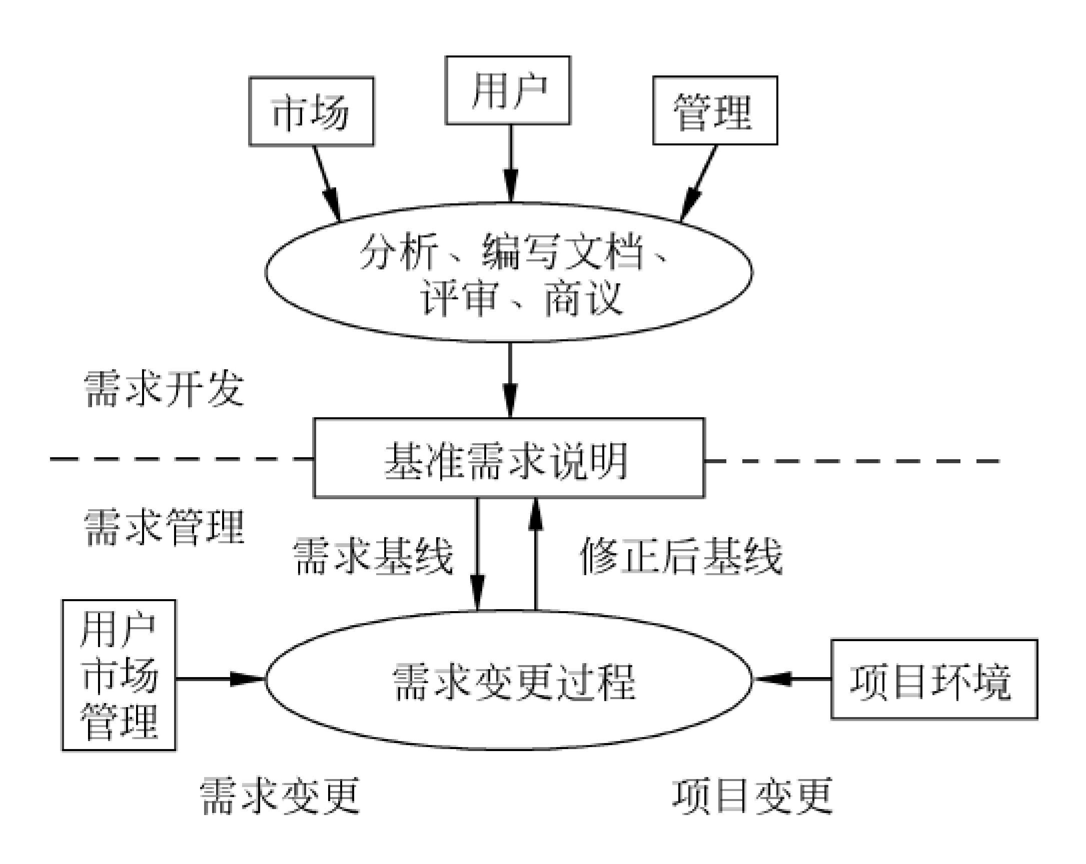


图 4.16　需求开发与需求管理的界限

4.7.2　需求管理的主要活动

需求开发的结果是形成用户与开发人员双方均可理解的系统逻辑视图和物理视图，它连接需求开发和需求管理，作为需求管理的输入。需求管理的内容包括在工程进展过程中为保证需求集成性及精确性进行的所有活动，具体内容可用图 4.17 表示。

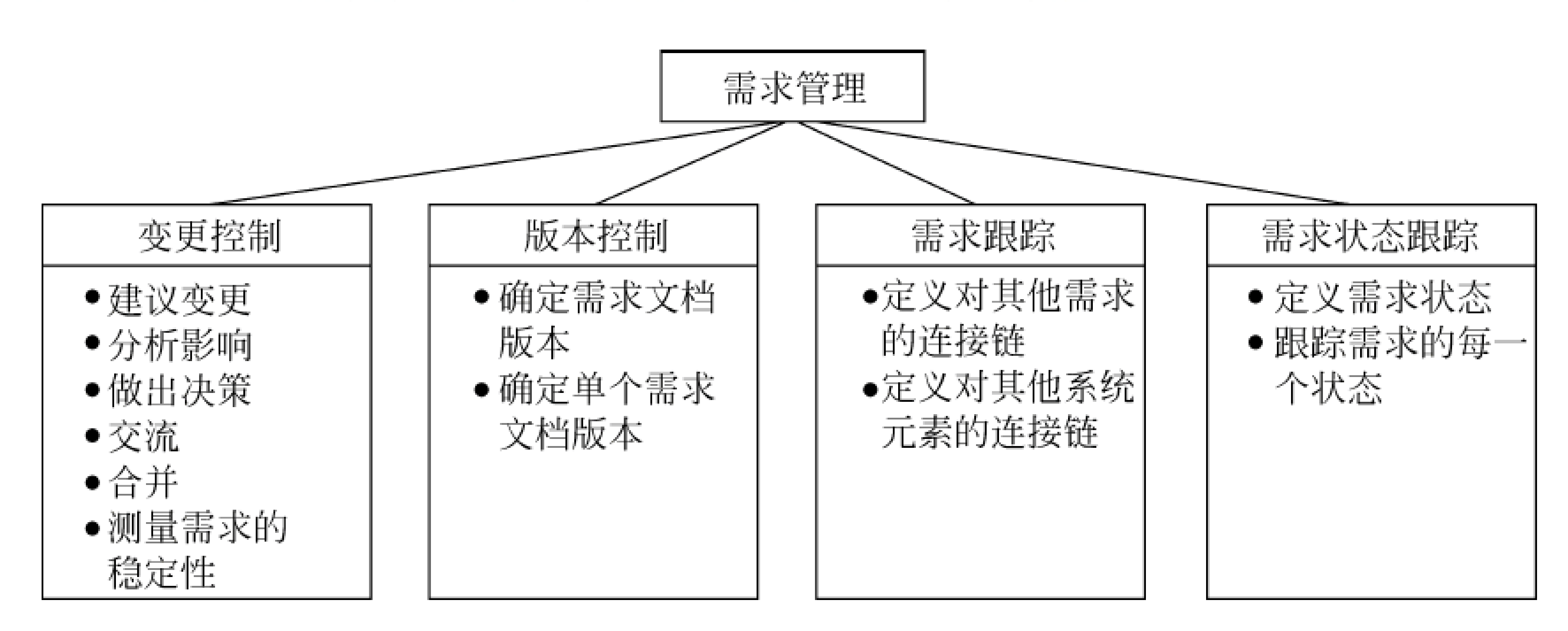


图 4.17　需求管理的主要活动

4.7.3　需求管理的方法与手段

需求管理的方法与手段主要有以下几个方面：

(1) 确定需求变更控制过程。确定一个选择、分析和决策需求变更的过程。所有的需求变更都需遵循此过程，商业化的问题跟踪工具都能支持变更控制过程。

(2) 建立变更控制委员会。组织一个由项目风险承担者组成的小组作为变更控制委员会，来确定进行哪些需求变更、变更是否在项目范围内、评估变更、对评估做出决策。确定选择哪些、放弃哪些，并设置实现的优先顺序，制定目标版本。

(3) 进行需求变更影响分析。评估选择的需求变更，确定变更对项目计划安排和其他需求的影响。明确与变更相关的任务并评估完成这些任务需要的工作量。通过这些分析将有助于变更控制委员会做出更好的决策。

(4) 跟踪所有受需求变更影响的工作产品。当进行某项需求变更时，参照需求跟踪能力矩阵找到相关的其他需求、设计模板、源代码和测试用例，这些相关部分可能也需要修改。这样能减少疏忽，使需求变更带来的产品变更相对容易。

(5) 建立需求基线和需求控制版本文档。确定一个需求基线，之后的需求变更遵循变更控制过程即可。每个版本的需求规格说明都必须是独立说明，以避免将底稿、基线或新旧版本相混淆。最好的办法是使用合适的配置管理工具在版本控制下为需求文档定位。

(6) 维护需求变更的历史记录。记录变更需求文档版本的日期和变更原因，以及由谁负责更新和更新的版本号等。版本控制工具能自动完成这些任务。

(7) 跟踪每项需求的状态。建立一个数据库，每条记录保存一项功能需求的重要属性和状态(已推荐的、已通过的、已实施的、已验证的)，这样在任何时候都能得到每个状态类的需求数量。

(8) 衡量需求稳定性。记录基本需求的数量和每周或每月的变更数量(添加、修改、删除)。过多的需求变更是一个"报警信号"，意味着问题并未真正弄清楚，项目范围并未很好地确定下来，或是政策变化较大。

(9) 使用需求管理工具。商业化的需求管理工具能在数据库中存储不同类型的需求，为每项需求确定属性，可跟踪其状态，并在需求与其他软件开发工作产品间建立跟踪能力联系链。

思考题

1. 理解软件需求各组成部分之间的关系。
2. 需求分析阶段的工作有何特点？
3. 理解需求分析的定义及需求分析的重要性。
4. 简述需求分析的实现步骤。
5. 获取用户需求的主要工作是什么？
6. 编写软件需求文档有哪些方法？
7. 需求分析评审的主要内容有哪些？
8. 需求分析的内容具体包括哪些？
9. 简述结构化分析的基本思想。
10. 简述用结构化分析方法进行需求分析的具体步骤。
11. 结构化分析方法的优缺点是什么？
12. 理解使用原型的目的和原型建立技术。
13. 使用用例表示软件需求的原因是什么？
14. 需求变更的原因主要有哪些？如何应对需求变更？
15. 需求分析验证常用的方法有哪些？
16. 简述需求分析验证的具体内容。
17. 需求管理的主要活动有哪些？
18. 需求管理的方法与手段有哪些？

第5章 概要设计

概要设计是需求分析的下一步工作，主要任务是把需求分析得到的DFD转换为软件结构。具体任务是：将一个复杂系统按功能进行模块划分，建立模块的层次结构及调用关系，确定模块间的接口及人机界面等。概要设计建立的是目标系统的逻辑模型，与计算机无关。

5.1 软件设计概述

软件开发过程在经过了需求分析阶段后，已经清楚了用户的需求，也就是已经解决了待开发软件"做什么"的问题，并且这些确定了的需求应该在软件需求规格说明中得到详细的叙述和充分的表达。进入软件设计阶段后，便可以开始研究软件需求的实施工作，即着手解决"如何做"的问题。

软件设计是把软件需求变换成为软件表示的过程。根据用数据、功能和行为模型表示的软件需求，采用相应的设计方法进行概要设计、详细设计、接口设计、数据库设计。需求分析与软件设计的关系如图5.1所示。

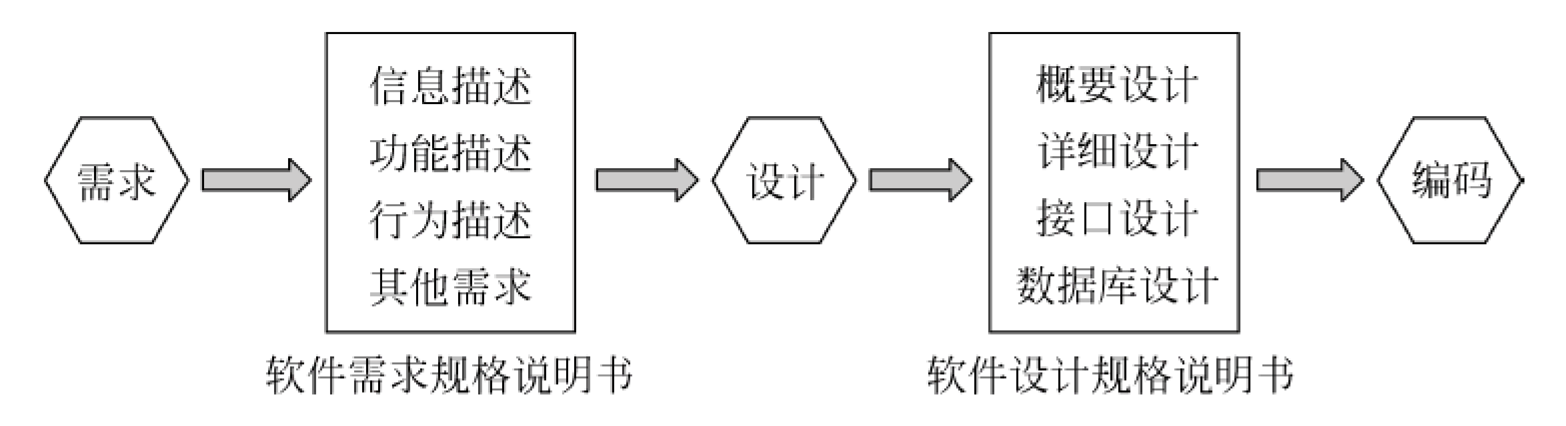


图5.1 需求分析与软件设计的关系

软件设计阶段各步骤的工作描述如下：

(1) 概要设计。根据数据流图，确定软件系统各主要组成部分之间的关系。

(2) 详细设计。根据控制规格说明、状态转换图和加工规格说明，将软件体系结构的组成部分转换成为软件组成部分的过程性描述。

(3) 接口设计。根据数据流图，定义软件内部各组成部分之间、软件与其他协同系统之间及软件与用户之间的交互方式。

(4) 数据库设计。将数据对象描述中的数据、实体联系图中描述的数据对象和关系以及数据字典中描述的详细数据内容，变换成为实现软件需求的数据结构。

软件设计是开发阶段的重要步骤，是后续软件开发和软件维护的基础。如果不进行软

件设计，只能建立一个不稳定的系统，如图 5.2 所示，只要出现一些微小的变动，就会导致软件崩溃，而且难以测试和维护。

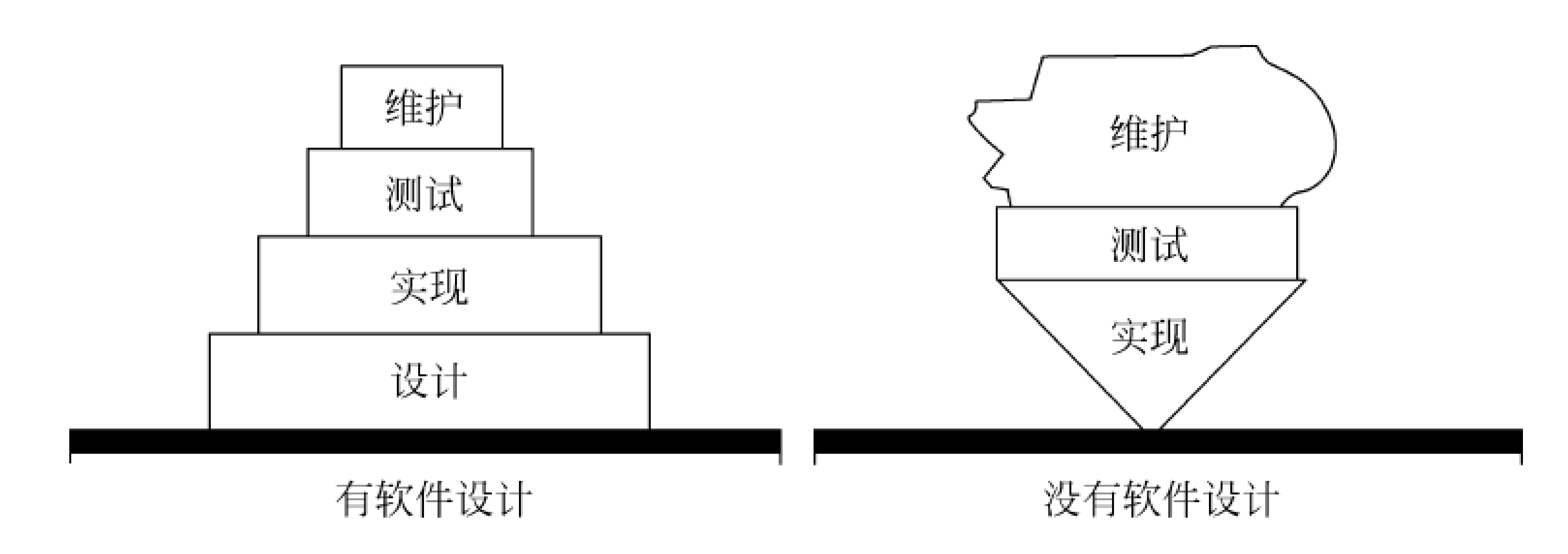


图 5.2　软件设计的重要性

从工程管理的角度看，软件设计通常分为两步进行：首先进行概要设计（也称为总体设计或结构设计），将软件需求转化为数据结构和软件系统结构，并建立各组成部分之间的接口；然后进行详细设计，通过对结构表示进行细化，得到软件的详细数据结构和算法。

软件设计过程的目标是：

（1）设计必须实现分析模型中描述的所有显式需求，必须满足用户希望的所有隐式需求。

（2）设计必须是可读、可理解、易于编程实现、易于测试、易于维护。

（3）设计应从实现的角度出发，给出与数据、功能和行为相关的软件全貌。

5.2　概要设计主要内容

5.2.1　设计任务

在 GB/T 8566—2007《信息技术 软件生存周期过程》中，有关概要设计的任务描述如下：

（1）应把软件项的需求转变为一种体系结构，该体系结构描述其顶层结构并标识各个软件部件。应确保软件项的所有需求都被分配给软件部件，并得到进一步的细化以便于详细设计。软件项的体系结构应形成文档。

（2）应编制关于软件项的外部接口，以及软件项各个软件部件之间接口的顶层设计，并形成文档。

（3）应编制数据库的顶层设计，并形成文档。

（4）应编制用户文档的最初版本，并形成文档。

（5）应确定软件集成的初步测试需求和进度安排，并形成文档。

（6）应根据评价准则评价软件项的体系结构、接口和数据库设计，评价结果应形成文档。评价准则包括软件需求的可追踪性、与结构设计的外部一致性、软件部件和软件单元之间的内部一致性、应用的设计方法和标准的适宜性、测试的可行性、运行与维护的可行性。

（7）应实施联合评审。

5.2.2　设计原则

概要设计应遵循抽象、逐步求精、模块化和信息隐蔽等原则。

1. 抽象

抽象是人类在认识世界和描述世界过程中常用的思维方法。人们在实践中认识到，现实世界中的一定事物、状态或过程之间总存在着某些相似的方面(即共性)。抽象就是把这些相似的方面集中和概括起来，暂时忽略它们之间的差异。也就是说，抽象就是抽出事物的本质特征，而暂时不考虑细节。

对软件进行模块化设计可以有不同的抽象层次。在最高的抽象层次上，可以使用问题所处环境的语言描述方法。而在较低的抽象层次上，则采用过程化方法。在软件设计过程中，常用的抽象方法有过程抽象、数据抽象和控制抽象三种。

(1) 过程抽象是对软件要执行的动作进行抽象。软件工程过程的每一步(或阶段)都是对软件解决方法中某个抽象层次的一次细化。例如，在需求分析和设计阶段，软件被抽象为系统的一个完整部件；在需求分析阶段，软件解决方法是使用在问题环境内熟悉的方式来描述，可被抽象为某个加工序列；在概要设计阶段，软件可被抽象为一个模块的层次结构；在详细设计阶段，软件可被抽象为程序流程图；在软件实现阶段，软件可被抽象为某个命名的指令序列；在源程序生成时，便达到了抽象的最低层次。

(2) 数据抽象是通过选择特定的数据类型及其相关功能特性的办法，仅仅保持抽取数据的本质特性所得到的结果，从而使其与细节部分的表现方式分开或把它们隐藏起来。数据抽象与过程抽象一样，允许设计人员在不同层次上描述数据对象的细节。例如，可以定义一个 shape 数据对象，并将它规定为一个抽象数据类型，用它的构成元素(triangle、circle、rectangle、polygon 等)来定义内部细节。此时，数据抽象 shape 本身由另外一些数据抽象构成。在定义了 shape 对象的数据类型之后，就可以引用它来定义其他数据对象，而不必涉及 shape 的内部细节。

(3) 控制抽象与过程抽象和数据抽象一样，可以包含一个程序控制机制而无须规定内部细节。控制抽象的一个例子是在操作系统中用来协调某些活动的同步信号。

2. 逐步求精

逐步求精是一种自顶向下的设计策略，由 Niklaus Wirth 首先提出，将软件体系结构按照自顶向下的方式，对各个层次的过程细节和数据细节逐步求精，直到能够用程序设计语言的语句实现为止，最终确立整个软件的体系结构。最初的说明只是概念性地描述了系统的功能或信息，并未提供有关功能的内部实现机制或内部结构的任何信息。设计人员对初始说明仔细推敲，进行功能细化或信息细化，给出实现细节，划分出若干成分，然后再对这些成分进行细化。随着细化工作的逐步进行，设计人员就能得到越来越多的细节。

抽象和逐步求精是两个互补的概念。抽象使设计人员忽略低层的细节，重点描述结构、过程和数据。逐步求精有助于设计人员在设计过程中揭示低层的细节。抽象和逐步求精能够帮助设计人员在设计工作中逐步建立完整的设计模型。

3. 模块化

模块是程序中能够逻辑分开的部分。实际上，模块是有一定功能、可以单独命名的被访问的数据和程序语句的集合。模块性是指软件由若干离散部分组成的离散程度，即软件模块化的程度(表明改变一个组成部分时对其他组成部分有多大影响)。

如果划分的模块是相互独立的,模块变得越小,每个模块所花费的工作量就越少。但是当分解的模块数增加时,模块间的联系加大,把这些模块连接起来的工作量也随之增加。因此,存在模块数量 M,使得总开发成本达到最小。模块规模、数量与费用的关系如图 5.3 所示。

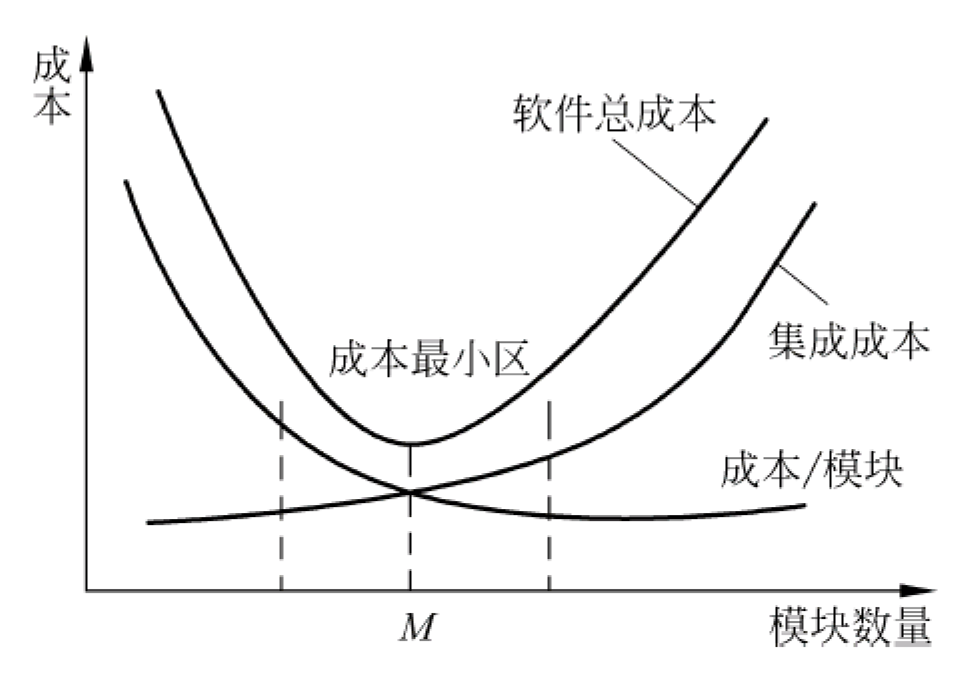


图 5.3 模块规模、数量与费用的关系

4. 信息隐蔽

信息隐蔽是指每个模块的实现细节对于其他模块来说是隐蔽的。也就是说,包含在一个模块内的信息(包括过程和数据)不允许其他不需要这些信息的模块使用。

通常,有效的模块化可以通过定义一组互相独立的模块实现,这些模块相互间的通信仅仅使用对于实现软件功能来说是必要的信息。通过抽象,可以确定组成软件的过程(或信息)实体,而通过信息隐蔽则可以定义和实施对模块过程细节和局部数据结构的存取限制。

采用信息隐蔽原则设计软件,为测试和维护期间的软件修改工作提供了极大便利。因为大多数数据和过程对软件的其他部分是隐蔽的,所以修改时被引入的一些未被察觉的错误很少能够传播到软件其他部位。

模块独立性是抽象、模块性和信息隐蔽等概念的直接产物。模块独立性是指软件系统中每个模块具有单一的功能,并与其他模块没有太多联系。即软件系统中每个模块只涉及软件要求的某个特定子功能,而与软件系统中其他模块的接口是简单的。一般采用两个准则(即内聚度和耦合度)来度量模块独立性。

内聚度是单个程序模块所执行的各个任务在功能上互相关联的程度。内聚度由高到低分为七种,其顺序如下。

(1) 功能内聚度:一个模块执行一个单一的、独立的功能。

(2) 顺序内聚度:如果一个模块有若干工作单元,它们都与同一功能紧密联系,又必须顺序执行。

(3) 信息内聚度:所有工作单元都集中于一个数据结构的同一区域。

(4) 过程内聚度:各工作单元间有一定关系,且必须按规定次序执行。

(5) 时间内聚度:一个模块要完成几个任务,这些任务要在同一时间段内执行。

(6) 逻辑内聚度:一个模块执行几个在逻辑上互相关联的任务。

(7) 偶然内聚度:一个模块执行几个在逻辑上几乎没有关系的任务。

耦合度是计算机程序中模块之间相互依赖的程度。耦合度由低到高分为七种,其顺序如下。

(1) 非直接耦合度:模块间没有直接联系,它们之间的联系完全通过主模块的控制和调用来实现。

(2) 数据耦合度:模块间仅用参数调用。

(3) 标记耦合度:模块接口传递数据结构的某一部分。

(4) 控制耦合度:模块间传递控制信息。

(5) 外部耦合度:模块通过外部环境相联系。

(6) 公共耦合度:多个模块引用同一个全局数据。

(7) 内容耦合度：一个模块要使用另一个模块内的数据或控制信息。

5.2.3 图形工具

概要设计通常用层次图、HIPO 图、结构图等图形工具描绘软件的组成结构。

1. 层次图

层次图用来描绘软件的层次结构。层次图中一个矩形框代表一个模块，矩形框间的连线表示调用关系，位于上方的矩形框代表的模块调用位于下方的矩形框代表的模块。层次图结构如图 5.4 所示，最顶层的矩形框代表系统的主控模块，主控模块通过调用第一层模块完成全部功能，第一层模块通过调用第二层模块完成功能，依此类推。

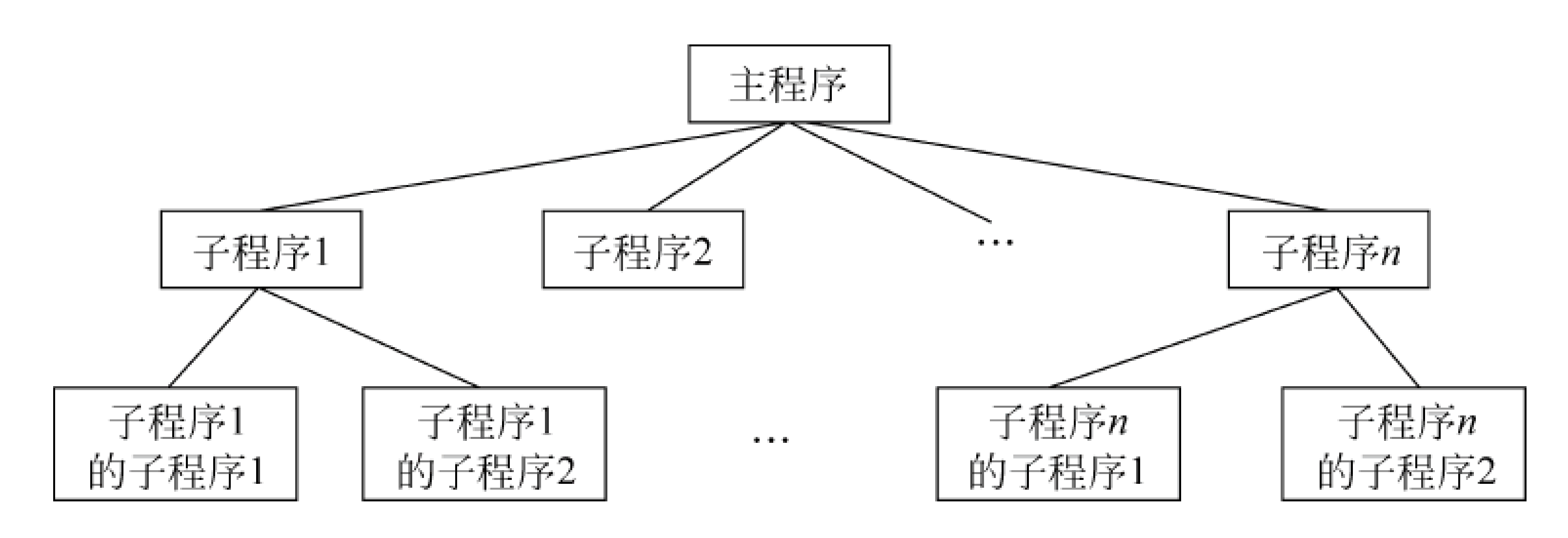


图 5.4 层次图

2. HIPO 图

HIPO(Hieraychy Input Process Output)图是由美国 IBM 公司发明的“层次图＋输入/处理/输出图”的英文缩写，也称为层次输入处理输出图。为了使层次图具有可追踪性，在层次图里除顶层的矩形框之外，为每个矩形框都加入了编号。编号按“X. Y. Z. N”的形式表示，“X”表示最高层模块，“X. Y”表示模块 X 的第 Y 个子模块，依此类推。HIPO 图结构如图 5.5 所示。

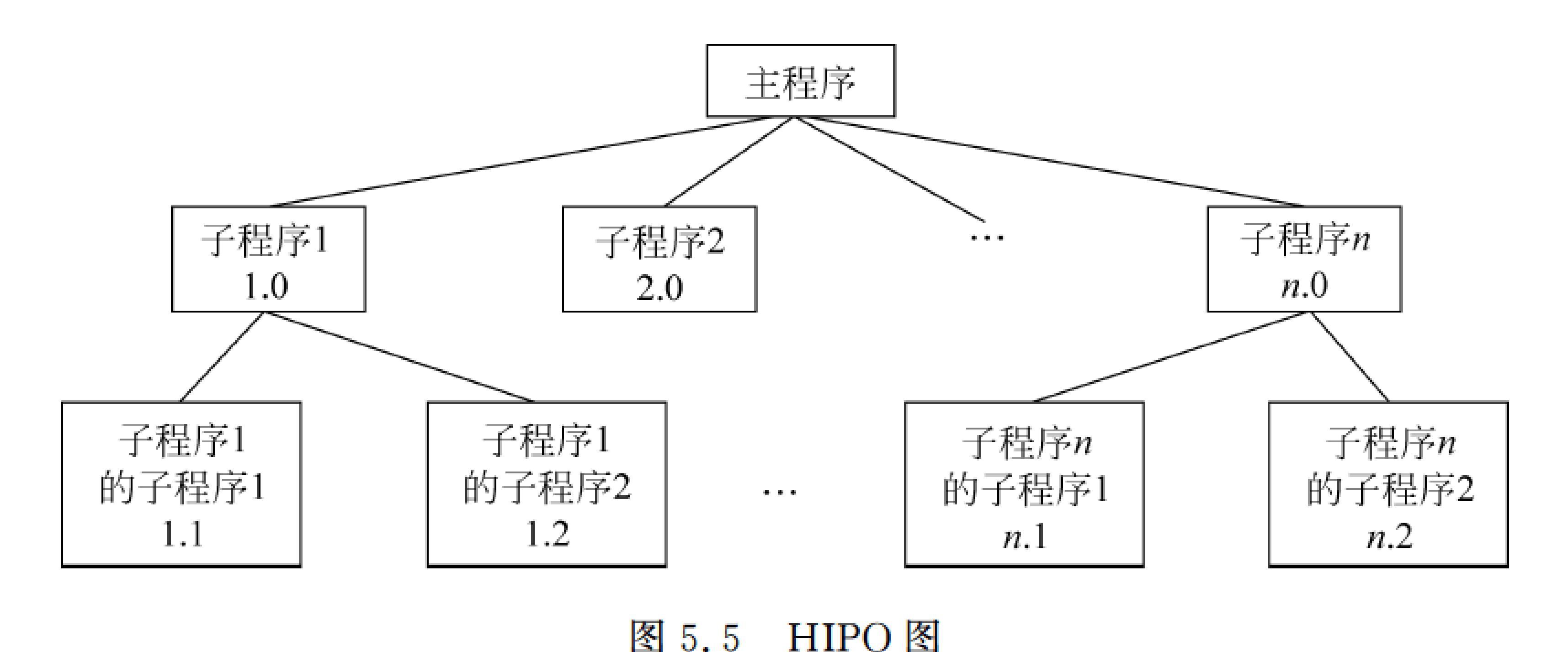
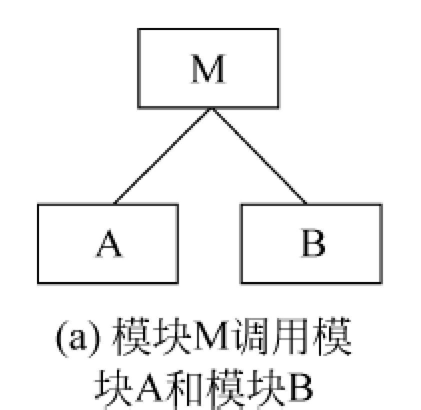


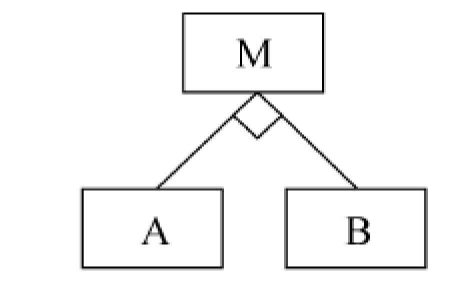
图 5.5 HIPO 图

3. 结构图

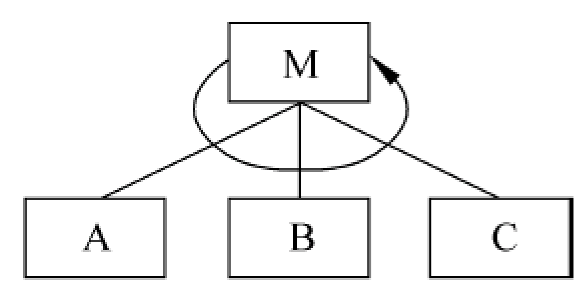
结构图是由 Yourdon 提出的，和层次图类似，也是描绘软件结构的图形工具。图中一个矩形框代表一个模块，框内注明模块的名称或主要功能，矩形框之间的箭头表示模块的调用关系。因为按照习惯，图中总是位于上方的矩形框代表的模块调用位于下方的矩形框代表的模块，即使不用箭头也不会产生二义性，为了简单，可以只用直线而不用箭头表示模块间的调用关系。结构图如图 5.6 所示。



(a) 模块M调用模块A和模块B



(b) 判定为真时，模块M调用模块A；判定为假时，模块M调用模块B



(c) 模块M循环调用模块A、模块B、模块C

图5.6 结构图

5.2.4 设计方法

概要设计通常采用结构化设计(Structure Design，SD)方法，也称为面向数据流的设计方法，是由Yourdon和Constantine等人于1974年提出的，与结构化分析(SA)方法相衔接，通过对数据流的分析来设计软件结构。SD方法对那些顺序处理信息且不含层次数据结构的系统最为有效，例如过程控制、复杂的数值分析过程以及科学与工程方面的应用。当SD方法用于完全的数据处理时，即使系统中使用层次数据也同样行之有效。

1. 数据流及数据流方法的设计过程

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。信息流有变换流和事务流两种类型，但很多时候数据流图又是由变换流和事务流组成的混合型数据流图。

1）变换流

根据基本系统模型，信息通常以“外部世界”的形式进入软件系统，经过加工处理以后再以“外部世界”的形式离开系统。如图5.7所示，信息沿输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流就叫做变换流。

2）事务流

基本系统模型意味着变换流，因此，原则上所有信息流都可以归结为这一类。但是当数据流图具有如图5.8所示的类似形状时，这种数据流是“以事务为中心的”，也就是说，数据沿输入通路到达一个处理，这个处理根据输入数据的类型，在若干个动作序列中选出一个来执行。这类数据流应该划为一类特殊的数据流，称为事务流。图5.8中的处理T称为事务中心，完成下述任务：

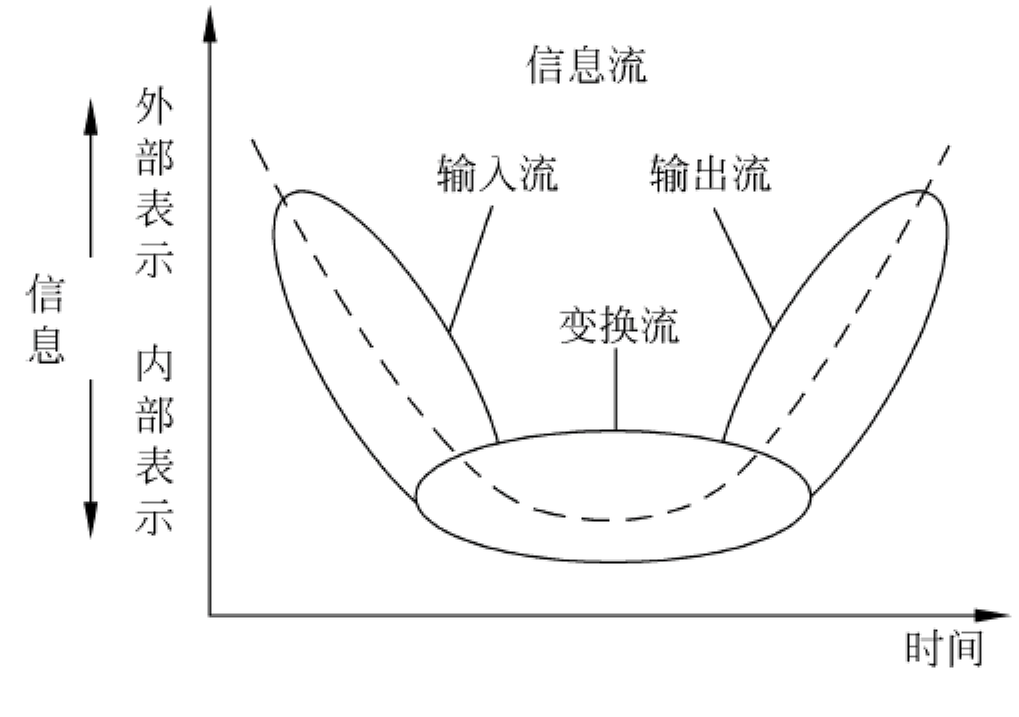


图5.7 变换流

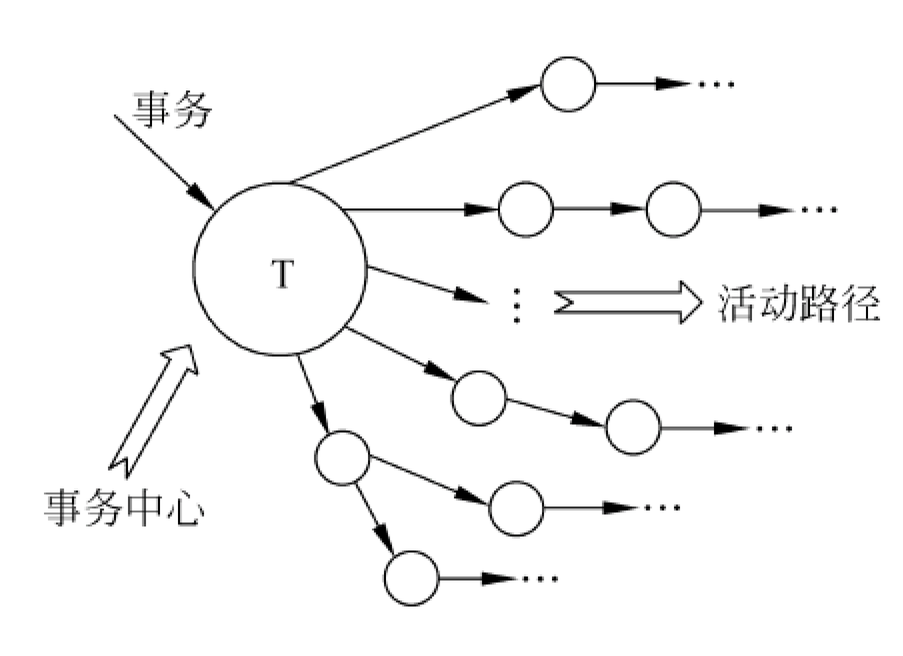


图5.8 事务流

(1) 接收输入数据(输入数据又称为事务)。

(2) 分析每个事务以确定它的类型。

(3) 根据事务类型选取一条活动通路。

3) 混合型数据流图

实际上所有的数据流图都是变换流,事务流是变换流的一种特殊形式。大多数系统的数据流图中,事务流和变换流往往交织在一起。混合型数据流图示例如图 5.9 所示。

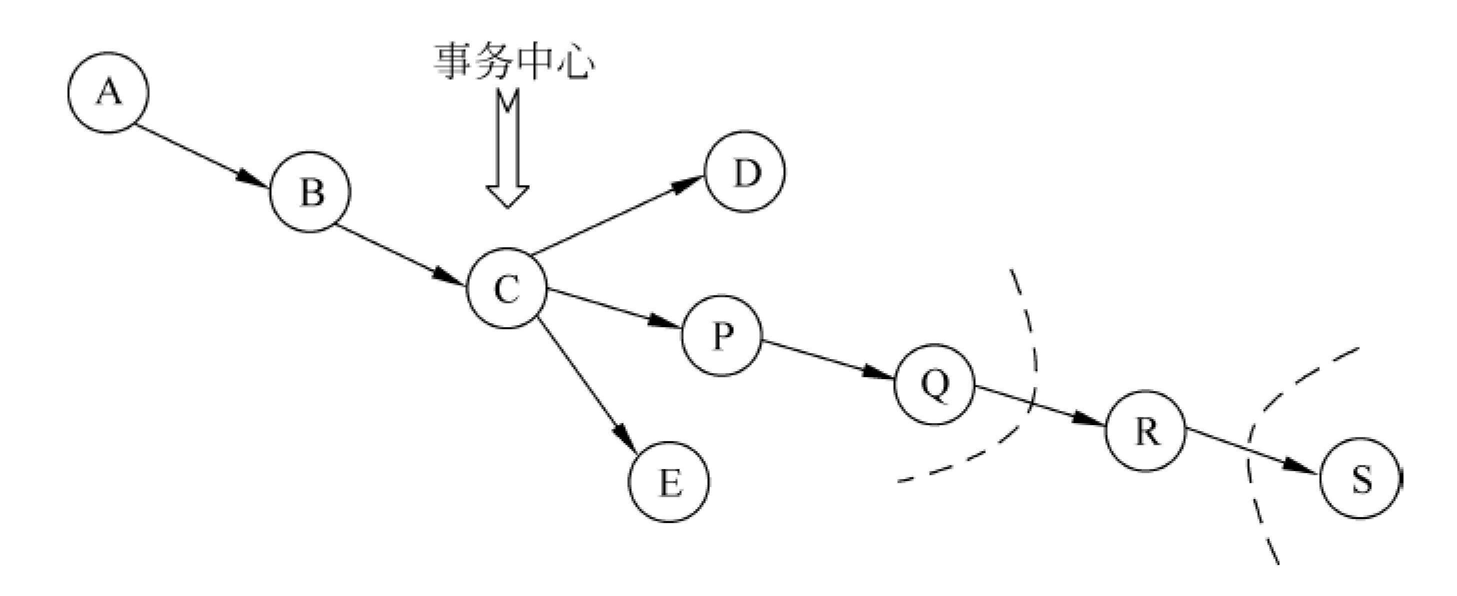


图 5.9 混合型数据流图示例

4) 设计过程

面向数据流方法的设计过程如图 5.10 所示。

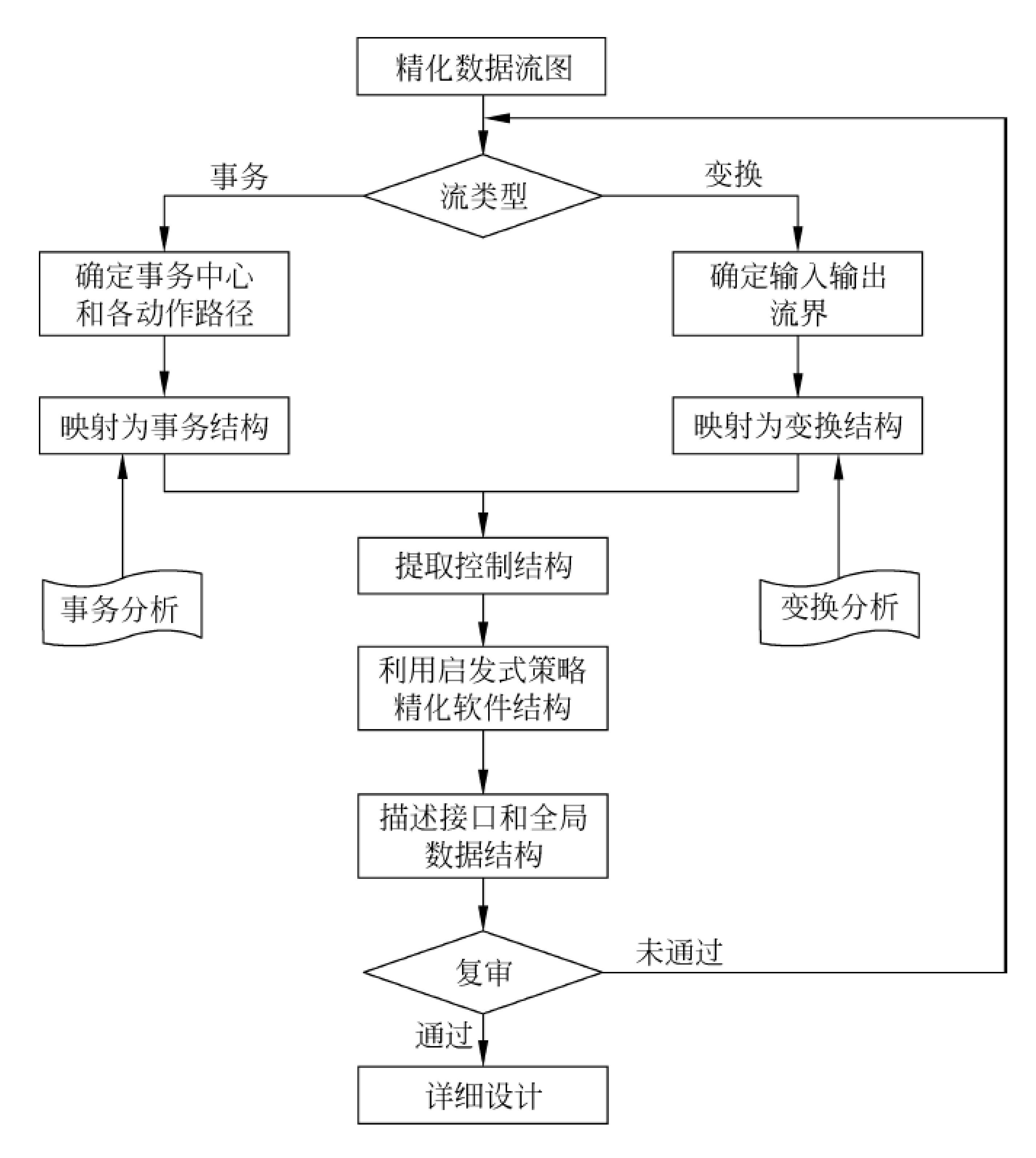


图 5.10 面向数据流的设计过程

2. 变换分析方法

变换分析方法的基本思想如图 5.11 所示,描述如下:

(1) 通过一系列的设计步骤,将变换型的数据流图映射为软件结构。

(2) 输入:需求规格说明(数据流图、数据字典、小说明)。

(3) 输出:软件总体结构。

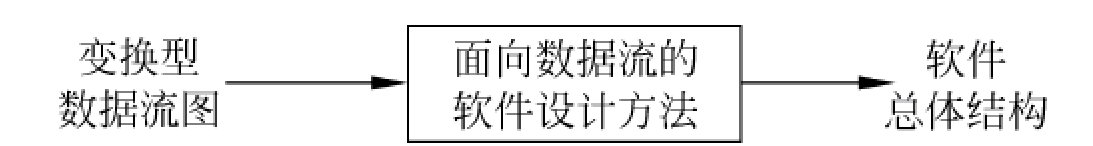


图 5.11　变换分析方法的基本思想

变换分析方法分为如下七个步骤。

步骤 1:复审基本系统模型。

基本系统模型是指顶级 DFD 和所有外部提供的信息。这一设计步骤是对系统规格说明书和软件需求规格说明书进行评估,这两个文档描述软件界面上信息的流程和结构。

步骤 2:复审和精化数据流图。

这一步主要是对软件需求规格说明书中的分析模型进行精化,确保 DFD 给出目标系统正确的逻辑模型,以获得足够详细的 DFD,确保 DFD 中每个转换代表一个规模适中、相对独立的子功能。

步骤 3:确定数据流图的类型。

信息流都可用变换流表示,但是如果有明显的事务流特征,则还应采用事务流的映射方法。设计人员负责判定在数据流图中占主导地位的信息流是变换流还是事务流。如果数据沿一个传入路径进来,沿多个传出路径离开,没有明显的事务中心,该信息流就属于变换流。

步骤 4:划分输入流和输出流边界。

划分输入流部分和输出流部分,孤立变换中心。划分因人而异,在一般情况下有些不同,但对整个目标软件系统影响不大。划分流界示例如图 5.12 所示。

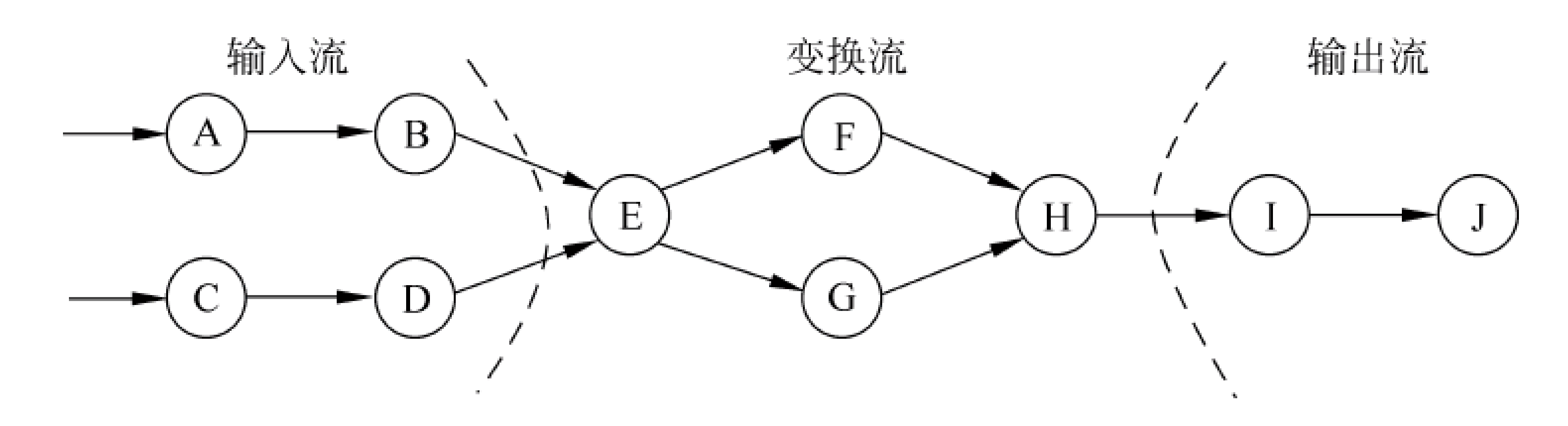


图 5.12　划分流界示例

步骤 5:执行一级分解。

(1) 任务:导出三个层次的软件结构。

① 底层模块:用于输入、输出和计算等基本功能。

② 中间层模块:协调、控制底层模块的工作。

③ 高层模块:用于协调和控制所有的从属模块。

(2) 原则:在确保完成系统功能,并保持低耦合度、高内聚度的前提下,尽可能减少模块数目。

(3) 表示方法：一级分解的结果可以用层次图、结构图等表示。

一级分解示例如图 5.13 所示。

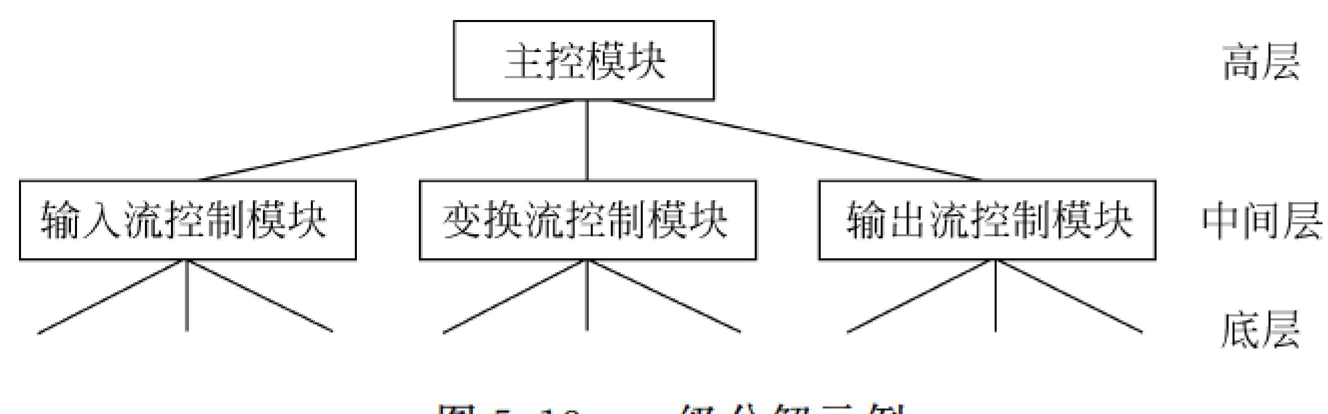


图 5.13　一级分解示例

步骤 6：执行二级分解。

(1) 任务：把 DFD 中每个变换映射为软件结构中的模块。

(2) 方法：从变换中心边界开始沿输入、输出通道向外移动，把输入、输出通道中的每个变换映射为软件结构中的一个模块；沿着输入流到输出流的方向移动，将每个变换映射为相应的模块。

(3) 有必要为每一模块书写简要处理说明，包括进出模块信息、模块处理功能陈述、有关限制和约束等。

二级分解示例如图 5.14 所示。

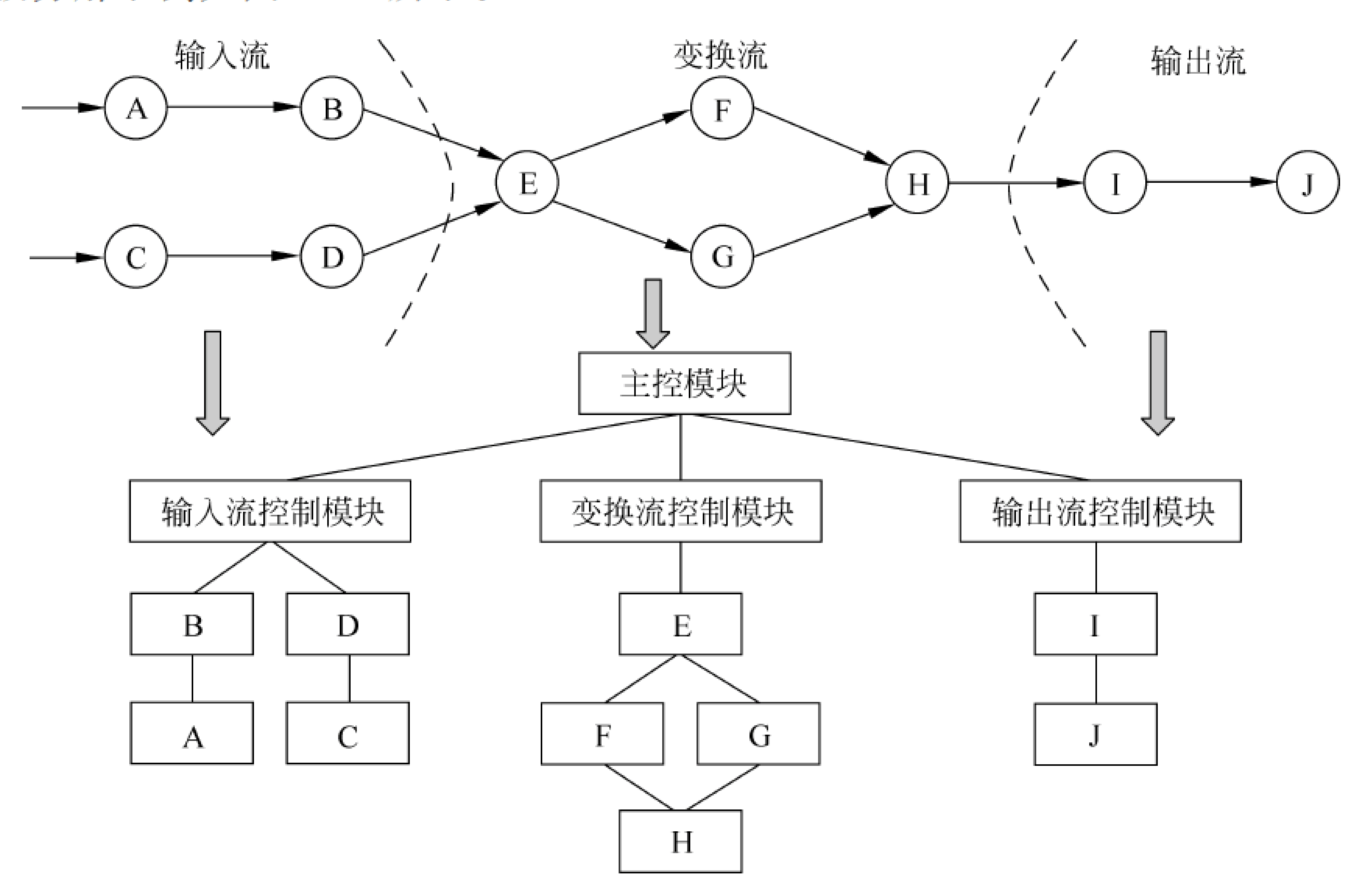


图 5.14　二级分解示例

步骤 7：精化软件结构，改良软件质量。

以"模块化"的思想，对软件结构中的模块进行拆并，以追求高内聚、低耦合、易实现、易测试、易维护的软件结构。精化软件结构示例如图 5.15 所示。

3. 事务分析方法

当数据流具有明显的事务特征时，应该采用事务流分析方法。事务流分析方法分为如下七个步骤。

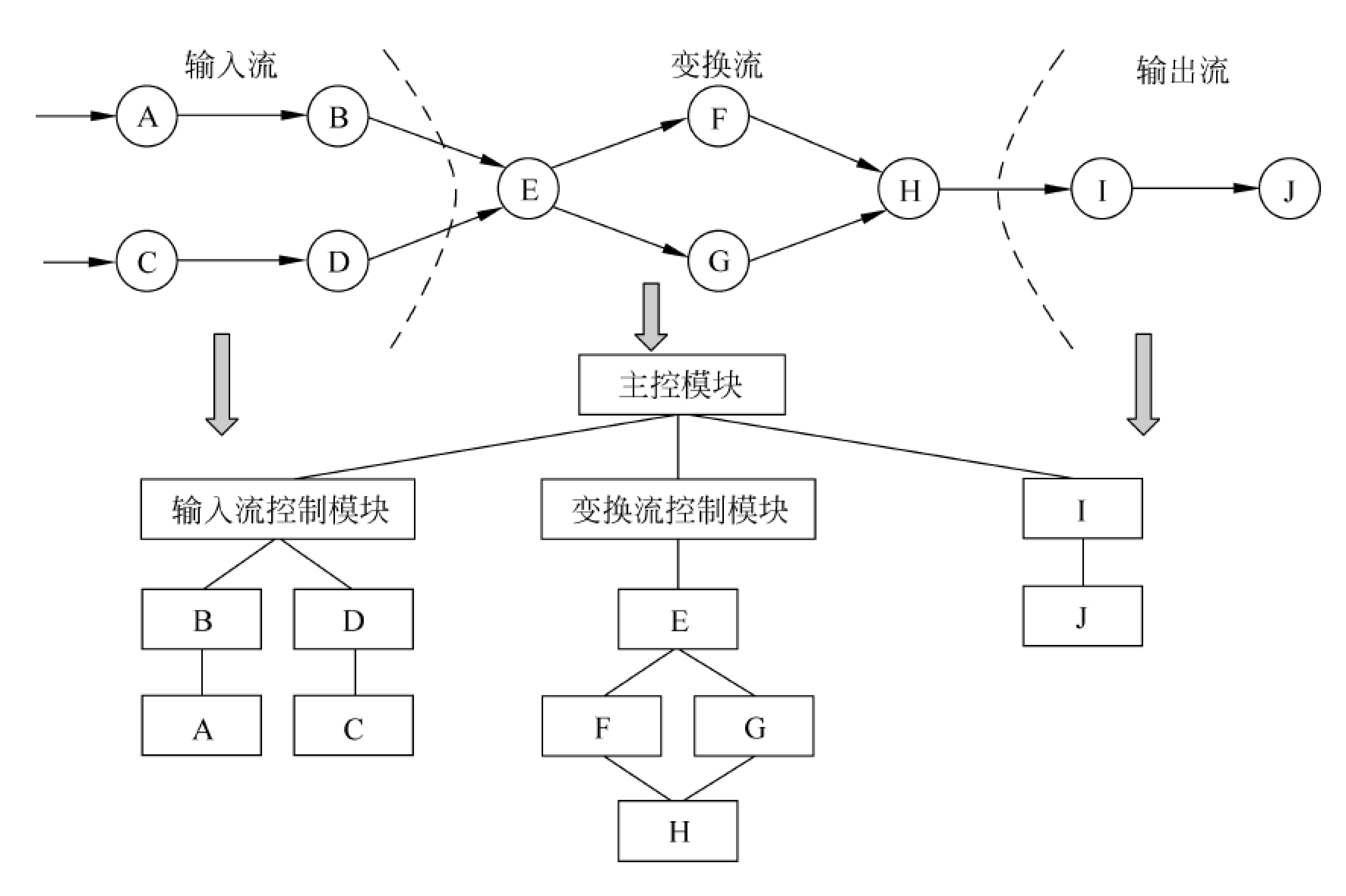


图 5.15　精化软件结构示例

步骤 1：复审基本系统模型。

步骤 2：复审和精化数据流图。

步骤 3：确定数据流图的类型。

步骤 4：识别事务流的各个组成部分。

(1) 把整个事务型 DFD 划分为以下三个部分：

① 接受路径部分。

② 事务处理中心。

③ 动作路径部分。

(2) 判定在每一条动作路径上数据流的特征：变换流或者事务流。

事务型 DFD 划分示例如图 5.16 所示。

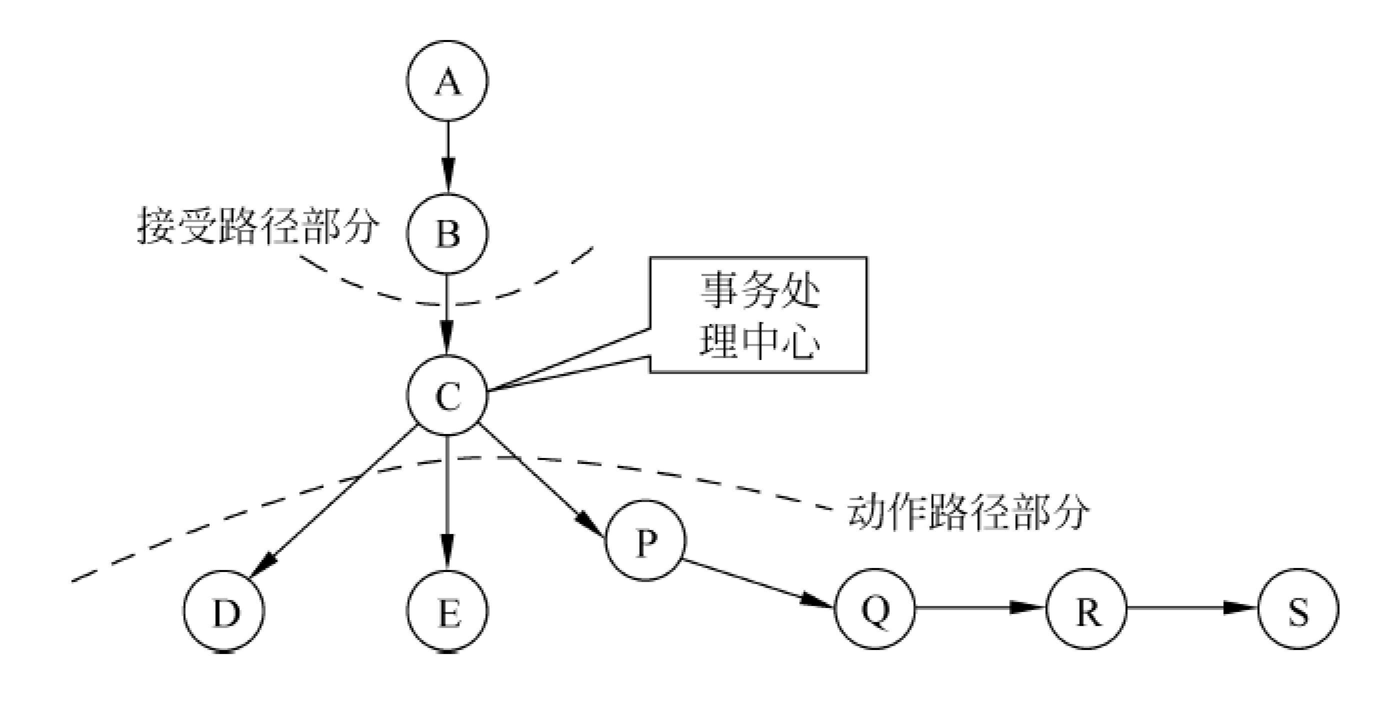


图 5.16　事务型 DFD 划分示例

步骤 5：把事务型 DFD 映射为软件结构。

由事务型 DFD 映射成的软件结构包括接受路径部分和动作路径部分。映射接受路径分支结构的方法和变换分析方法相似，即从事务处理中心的边界开始，把沿着接受路径部分

的通路和处理映射成模块。动作路径部分通过加入一个调用模块，控制下层的所有活动模块；然后把数据流图中的每个活动流通路映射成与它的流特征相对应的结构。事务型 DFD 映射为软件结构示例如图 5.17 所示。

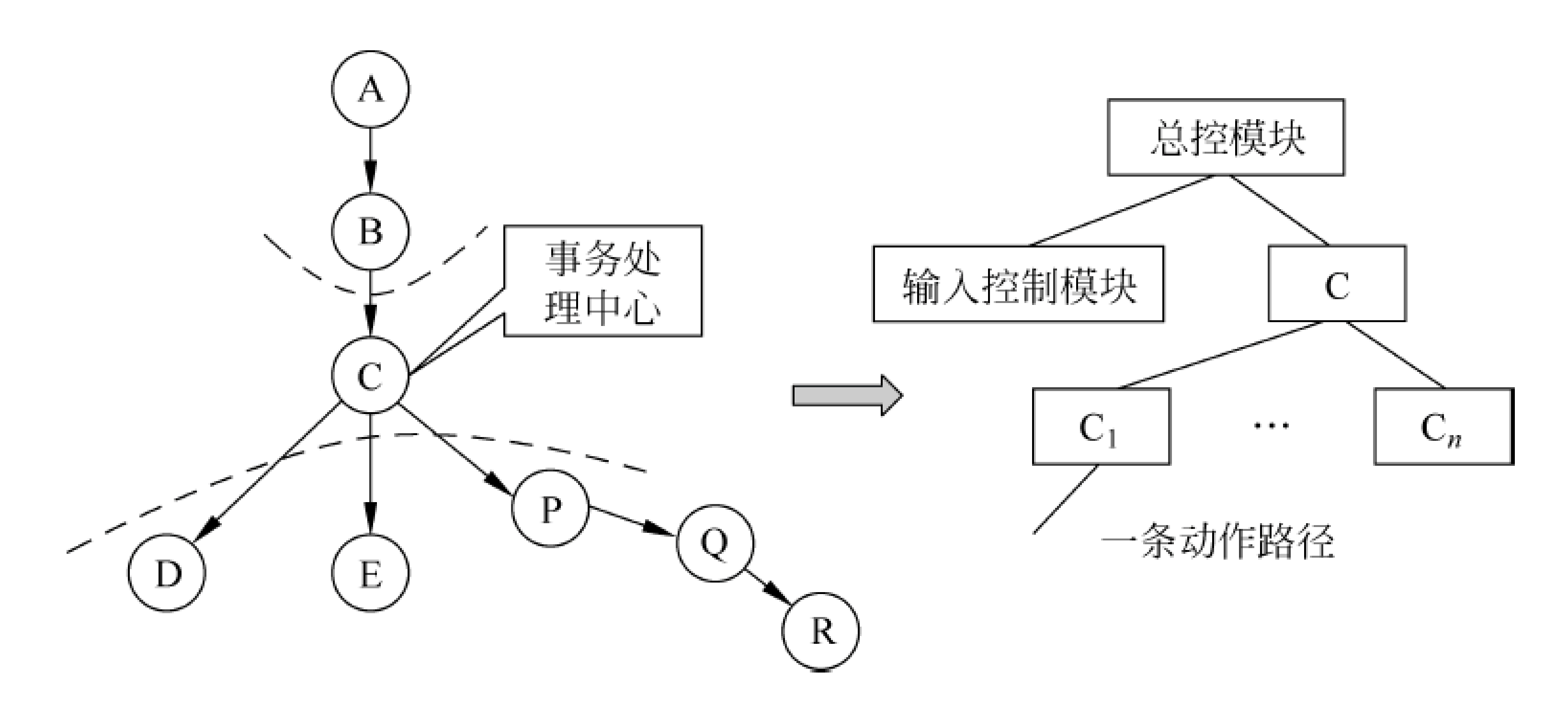


图 5.17 事务型 DFD 映射为软件结构示例

步骤 6：分解精化事务结构以及每个动作路径。

分解精化事务结构以及每个动作路径示例如图 5.18 所示。

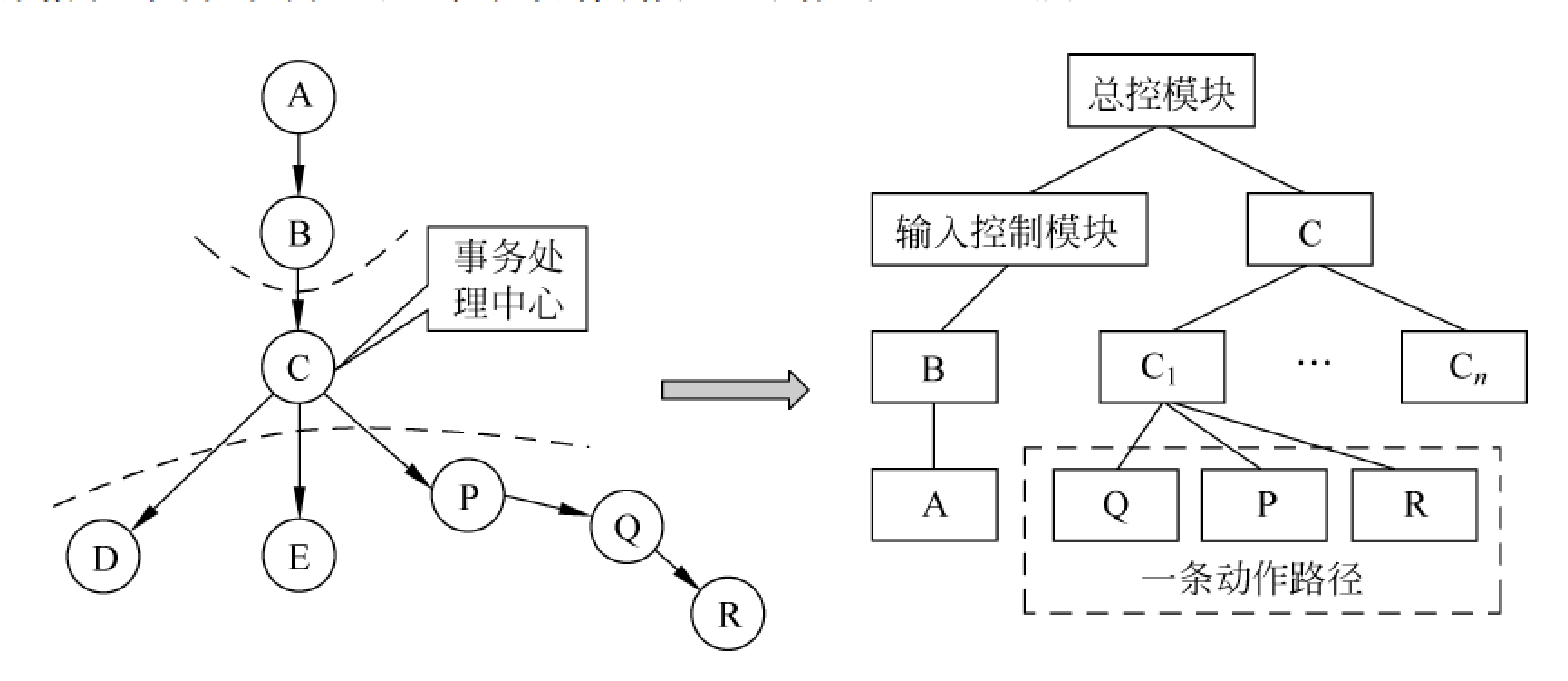


图 5.18 分解精化事务结构以及每个动作路径示例

步骤 7：精化初步软件结构。

精化初步软件结构示例如图 5.19 所示。

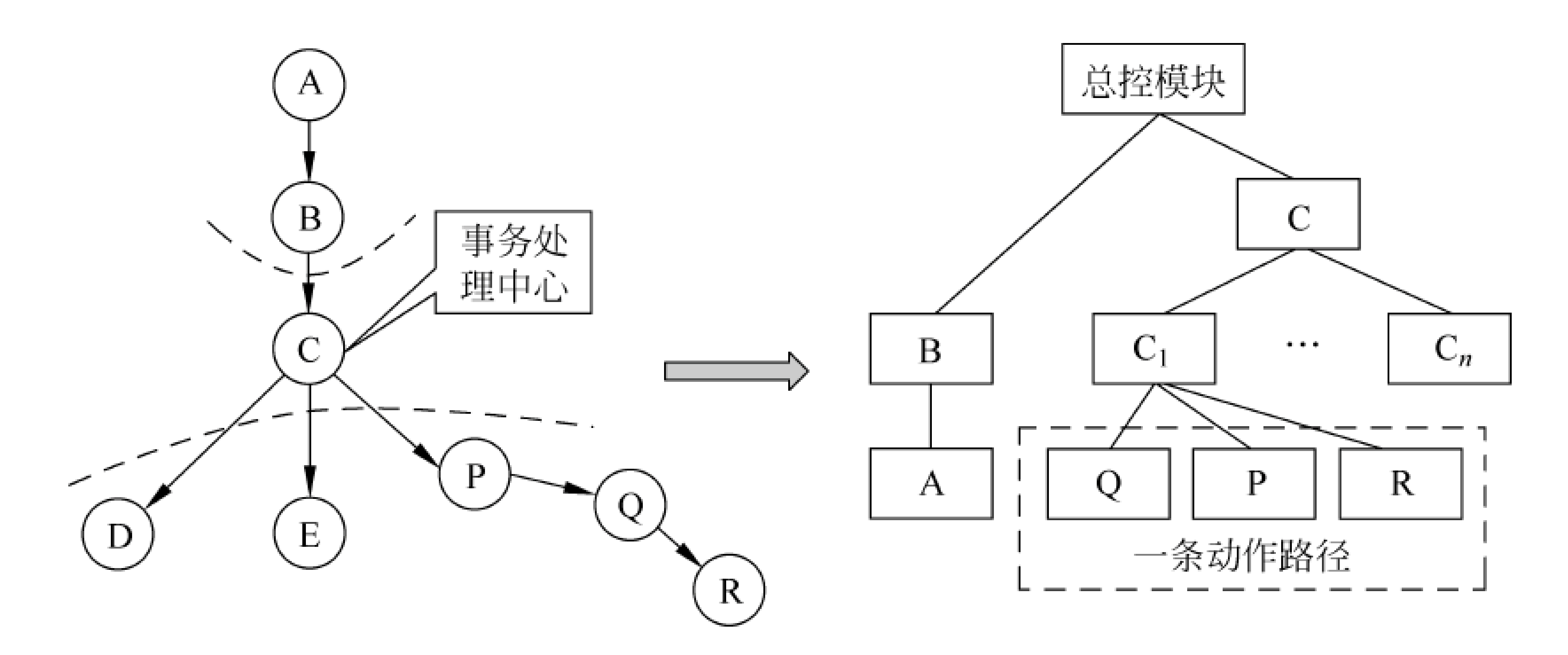


图 5.19 精化初步软件结构示例

5.2.5　启发式设计策略

启发式设计策略能给软件工程师有益的启示，帮助软件工程师找到改进软件设计、提高软件质量的途径。常用的启发式设计策略有以下几种：

(1) 改造软件结构，提高内聚度，降低耦合度。

如果在几个模块中发现共有的子功能，一般应将该子功能独立出来作为一个模块，以提高模块的内聚度，如图 5.20(a)所示。

合并模块通常是为了减少控制信息传递，以及对全局数据的引用，同时降低接口的复杂性。合并模块可以降低模块之间的耦合度，如图 5.20(b)所示。

模块的规模没有固定要求，以保持模块的独立性为原则。一般而言，模块规模在一页左右为宜(高级语言为 75 个语句左右)。

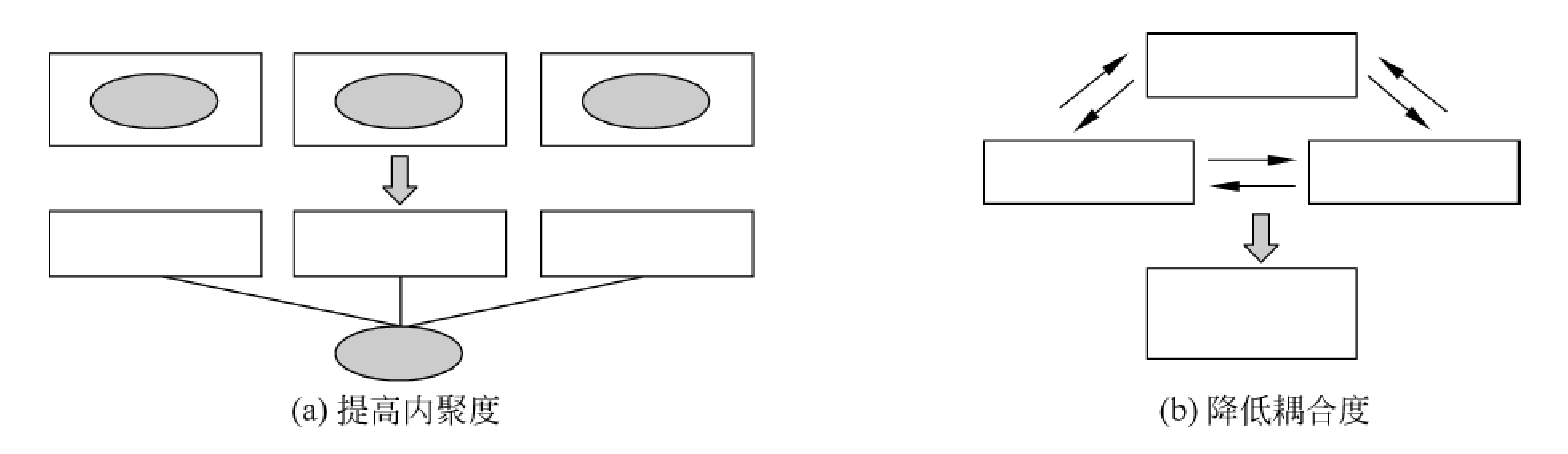


图 5.20　提高内聚度，降低耦合度

(2) 减少扇出，追求高扇入。

经验表明，设计良好的软件结构通常顶层扇出较高，中间层扇出较低，底层又高扇入到公共模块中去。应避免图 5.21 中的(a)结构，提倡图 5.21 中的(b)结构。

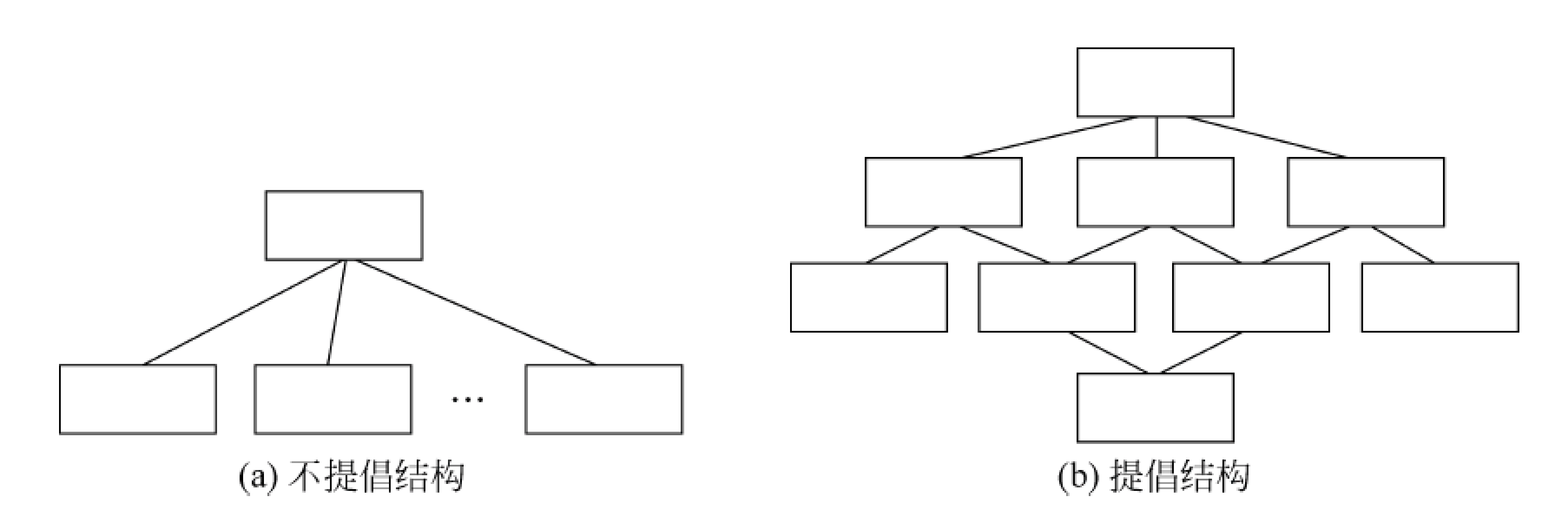


图 5.21　减少扇出，追求高扇入

(3) 使任意模块的作用域在其控制域内。

作用域是指受模块内部判定影响的所有模块，控制域是指其所有的下属模块。使任意模块的作用域在其控制域内示例如图 5.22 所示。

(4) 降低模块的接口复杂度和冗余度，提高协调性。

模块接口复杂是软件发生错误的一个主要原因，模块接口应尽可能简单并与模块功能相一致。接口复杂或不一致(即传递的数据之间没有联系)是高耦合或低内聚的表征，应重新分析模块的独立性。

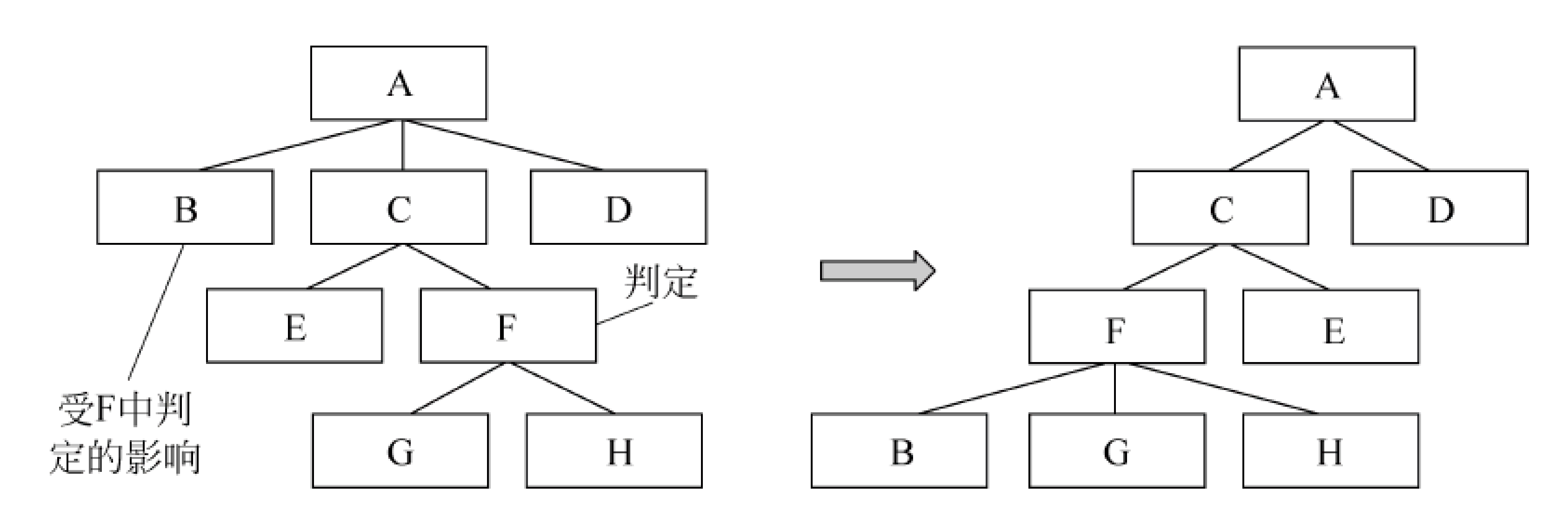


图 5.22 模块的作用域和控制域

(5) 模块功能可预测,避免对模块施加过多限制。

模块功能可预测是指输入恒定,则输出恒定,依据输入数据预测输出结果。另外,如果设计时对模块的局部数据结构、控制流程以及外部接口等因素限制过多,以后为去掉这些限制要增加维护开销。

(6) 追求单入口、单出口的模块。

当从顶部进入模块并且从底部退出时,软件是比较容易理解的,也是比较容易维护的。在设计时不要使模块间出现内容耦合。

(7) 为满足设计和可移植性要求,把某些软件用包封装起来。

软件设计常常附带一些特殊限制(例如要求程序采用覆盖技术)。此时,根据模块重要程度、被访问的频率及两次引用的间隔等因素对模块分组。

程序中那些供选择或"单调"的模块应单独存在,以便被高效率地加载。

5.3 接口设计

接口设计是概要设计和数据库设计的补充,从总体说明外部接口和内部接口。外部用户、软/硬件环境与软件系统的接口称为外部接口;已划分出的模块间接口称为内部接口。需求分析中已明确了软件系统与其他外围系统的接口,这里应将这些接口与划分出的具体模块相联系,对接口的命名、顺序、数据类型和传递形式等做出具体规定,也就是将这些接口分配到具体的模块中。接口设计的成果是《接口设计说明》。

在《接口设计说明》中,对所标识的每个接口,应陈述赋予该接口的项目唯一标识符,应使用名称、编号、版本和文档引用等标识接口实体(系统、配置项、用户等)。标识应说明哪些实体具有固定的接口特性,哪些实体正被开发或修改。适当时可使用一个或多个接口图来描述这些接口。

通过项目唯一标识符标识接口,简要地标识接口实体,并且根据需要分条描述接口实体单方或双方的接口特性。如果给定接口实体在本文档中没有提到,但是其接口特性需要在本文档描述的接口实体中提到,则这些特性应以假设的形式描述。《接口设计说明》应包含以下内容:

(1) 接口实体分配给接口的优先级别。

(2) 要实现的接口类型。例如实时数据传送、数据的存储和检索等。

(3) 接口实体必须提供存储、发送、访问、接收的单个数据元素的特性。例如名称、标识

符，数据类型，大小和格式，计量单位，范围或可能值的枚举，准确度和精度，优先级别、时序、频率、容量、序列、其他约束条件，保密性和私密性要求，来源和接收者等。

（4）接口实体必须提供存储、发送、访问、接收的数据元素集合体（记录、消息、文件、显示、报表等）的特性。例如名称、标识符，数据元素集合体中的数据元素及其结构，媒体和媒体中数据元素、数据元素集合体的结构，显示和其他输出的视听特性（例如颜色、布局、字体、图标和其他显示元素、蜂鸣、亮度等），数据元素集合体之间的关系，优先级别、时序、频率、容量、序列、其他约束条件，保密性和私密性要求，来源和接收者等。

（5）接口实体必须提供为接口使用通信方法的特性。例如项目唯一标识符，通信连接、带宽、频率、媒体及其特性，消息格式化，流控制，数据传送速率、周期性/非周期性、传输间隔，路由、寻址、命名约定，传输服务（包括优先级别和等级），安全性、保密性、私密性方面的考虑等。

（6）接口实体必须提供为接口使用协议的特性。例如项目唯一标识符，协议的优先级别、层次，分组（包括分段和重组、路由、寻址等）、合法性检查、错误控制和恢复过程，同步（包括连接的建立、维护、终止等），状态、标识、任何其他的报告特征等。

（7）其他所需的特性。例如接口实体的物理兼容性（尺寸、容限、负荷、电压和接插件兼容性等）。

5.4 概要设计与详细设计的衔接

概要设计就是根据目标系统的逻辑模型建立目标系统的物理模型，以及根据目标系统逻辑功能的要求，考虑实际情况，详细确定目标系统的结构和具体的实施方案。概要设计的目的是在保证实现逻辑模型的基础上，尽可能提高目标系统的简单性、可变性、一致性、完整性、可靠性、经济性、运行效率和安全性。

一个软件系统具有层次性（系统各组成部分的管辖范围）和过程性（处理动作的顺序）特征。在概要设计阶段，主要关心系统的层次结构；到了详细设计阶段，需要考虑系统的过程性，即“先干什么，后干什么”，以及系统各组成部分是如何联系在一起的。

软件系统是一个人机系统，计算机在人的参与下高效率地完成大量的处理工作。在实际应用中，总要有某种意义上的人工干预，这种干预总是在系统的关键部分，即控制与决策部分。若这部分也让计算机自动处理，就会降低效率，提高费用，甚至不可能处理。因此，在软件系统的关键部分，少量的手工操作是不可避免的。在进行软件系统的详细设计时首先就要分清哪些工作由计算机来做，哪些工作由手工完成，从而确定模块的实现方式，并据此把模块分成不同类型，再按照不同类型用不同的方法来设计实现方案。

每个模块均有各自要完成的任务。这个任务是适于用计算机处理还是适于用人工处理，在设计时就必须做出选择。由于计算机处理和人工处理各有特点，如何合理分工，扬长避短，充分发挥组织现有的人、机资源的作用，使软件系统的功能得到最好的实现，这是设计人员必须解决的问题。因此，在进行模块实现的计算机处理与人工处理划分时应了解这两种处理过程的不同特点，结合模块实现的人机处理划分原则进行选择。

模块实现计算机处理与人工处理划分的一般原则是：

（1）对复杂的计算、大量重复的数学运算，如统计、汇总、分配等，对结构化程度高的数

据处理，如数据传送、存储、分类、检索、编制单证报表等，适合于计算机处理。

(2) 各种管理模型、高层次的数学模型，如运筹学、数理统计、预测等处理，数据量大、算法复杂，适合于计算机处理。

(3) 数据格式不固定，例外情况较多及需要经验来判断的工作，目前没有成熟的技术可以应用或者代价太高，适合于人工处理。

(4) 决策性问题，先由计算机处理提供尽可能多的资料，来辅助与支持人进行最后的决策。

思考题

1. 理解需求分析与软件设计的关系。
2. 简述软件设计及各阶段的工作。
3. 软件设计的目标是什么？
4. 简述概要设计的任务及原则。
5. 理解模块规模、数量与费用的关系。
6. 在概要设计时如何把握内聚度与耦合度？
7. 掌握概要设计的层次图、HIPO图、结构图等图形工具。
8. 在概要设计时如何区分信息流是变换流还是事务流？
9. 掌握变换分析方法与事务分析方法的步骤。
10. 常用的启发式设计策略有哪些？
11. 如何理解接口设计？《接口设计说明》文档包括哪些内容？
12. 概要设计与详细设计如何衔接？

第6章 详细设计

概要设计确定了软件的总体结构，将软件划分为功能独立的若干模块，并描述每个模块的外部特性(功能与接口)，以及定义模块通用的全局数据结构和基本的用户界面风格。详细设计是对概要设计的进一步细化，确定模块的两个内部特性，即描述每个模块的执行过程(怎么做)和定义模块的局部数据结构。

6.1 设计任务

在 GB/T 8566—2007《信息技术 软件生存周期过程》中，有关详细设计的任务如下：

(1) 应对软件项的每一软件部件进行详细设计。软件部件应细化到更低级别，这些级别包含能被编码、编译、测试的软件单元。应确保来自这些软件部件的所有软件需求都被分配到软件单元。详细设计应形成文档。

(2) 应编制关于软件项外部接口、软件部件之间以及软件单元之间接口的详细设计，并形成文档。接口的详细设计应允许在不需要更多信息的情况下编码。

(3) 应编制数据库的详细设计并形成文档。

(4) 必要时，应更新用户文档。

(5) 应规定要测试的软件单元的测试需求和进度安排，并形成文档。测试需求应包括对软件单元在需求边界的强化要求。

(6) 应更新软件集成的测试需求和进度安排。

(7) 应根据评价准则评价软件详细设计和测试需求，并将评价结果形成文档。评价准则包括软件需求的可追踪性、软件部件和软件单元之间的内部一致性、所应用的设计方法和标准的适宜性、测试的可行性、运行和维护的可行性。

(8) 应实施联合评审。

6.2 结构程序设计

1. 结构程序设计的产生与发展

"结构程序设计"概念被称为软件发展中的第三个里程碑(第一、第二个里程碑分别是子程序和高级语言)，是由荷兰的计算机科学家 Edsgar Wybe Dijkstra(埃德斯加 · 狄克斯特

拉)提出的。在 1965 年召开的 IFIP 会议上,Dijkstra 提出:“Goto 语句可以从高级语言中取消,一个程序的质量与程序中所含的 Goto 语句的数量成反比。”1966 年,C • Bohm 和 G • Jacopini 证明了程序设计语言中,只要有顺序、选择和循环三种形式的控制结构,就足以表示出所有程序结构。

1968 年 3 月,ACM 通讯(*Communications of ACM*)登出了 Dijkstra 的那封影响深远的信《Goto 语句看来是有害的》(*Goto Statement Considered Harmful*),在信中他根据自己编程的实际经验和大量观察,得出如下结论:一个程序的易读性和易理解性,同其中包含的无条件转移控制的个数成反比关系,也就是说,转向语句的个数越多,程序就越难读、难懂。因此他认为“Goto 语句是有害的”,并从而启发了结构化程序设计的思想。1972 年,他与当时在爱尔兰昆士大学任教的英国计算机科学家、1980 年图灵奖获得者霍尔(C. A. R. Hoare)合著了《结构程序设计》(*Structured Programming*, Academic Pr.)一书,进一步发展并完善了这一思想,并且提出了另一个著名的论断:“程序测试只能用来证明有错,绝不能证明无错。”

有关 Goto 语句的争论,直到 1974 年克努特发表文章《带有 Goto 语句的结构化程序设计》之后才平息下来,他主张在语言控制划分中仍然保留 Goto 语句,在功能方面不加限制,但限制其使用范围。结构化程序允许有 Goto 语句,但它只能在本程序块内使用,不允许从一个结构转移到另一个结构。

2. 结构程序设计的基本控制结构

只要有顺序、选择、循环这三种基本结构,就能实现任何单入口、单出口的程序。基本控制结构如图 6.1 所示。

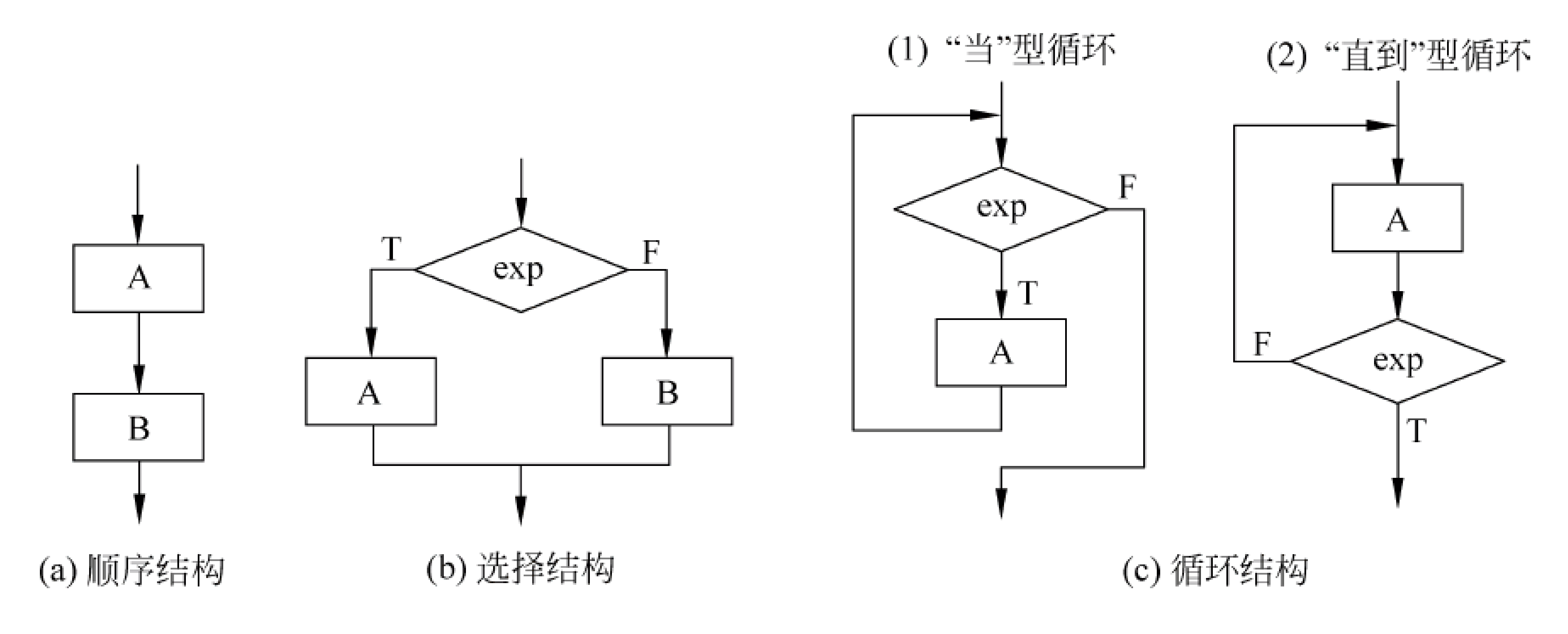


图 6.1 基本控制结构

图 6.1(a)是顺序结构,先执行 A 再执行 B。

图 6.1(b)是 If…Then…Else 型选择(分支)结构。

图 6.1(c)是循环结构。其中,(1)为“当”型循环(Do…While 结构),先判断后执行,当条件为真时执行;(2)为“直到”型循环,先执行后判断,直到条件为真时退出循环。

实际上,利用顺序结构和循环结构就完全可以实现选择结构,因此,理论上最基本的控制结构只有两种。

3. 结构程序设计的经典定义

结构程序设计的经典定义有以下两个：

(1)“如果一个程序的代码块仅仅通过顺序、选择和循环这三种基本控制结构进行连接，并且每个代码块只有一个入口和一个出口，则称这个程序是结构化的。”

(2)“结构程序设计是尽可能少用 Goto 语句的程序设计方法。最好仅在检测出错误时才使用 Goto 语句，而且应该总是使用前向 Goto 语句。”

4. 结构程序设计技术的优越性

结构程序设计技术的优越性包括以下几点：

(1) 自顶向下逐步求精的方法符合人类解决复杂问题的普遍规律，因此可以显著提高软件开发的成功率和生产率。

(2) 用先全局后局部、先整体后细节、先抽象后具体的逐步求精过程开发出的程序，有清晰的层次结构，因此容易阅读和理解。

(3) 不使用 Goto 语句，仅使用单入口、单出口的控制结构，使得程序的静态结构和动态执行情况比较一致，易于阅读和理解。

(4) 控制结构有确定的逻辑模式，编写程序代码只限于很少几种直截了当的方式，因此源程序清晰流畅。

(5) 程序清晰和模块化，使得在修改和重新设计一个软件时，可以重用的代码量最大。

(6) 程序的逻辑结构清晰，有利于程序的正确性证明。

6.3 表示工具

在详细设计阶段，要决定各个模块的实现算法，并精确地表达这些算法。在理想情况下，算法过程描述采用自然语言表达，这样对不熟悉软件的人员，要理解规格说明就比较容易，不需要重新学习。但是自然语言在语法和语义上往往具有多义性，常常要依赖上下文才能将问题表达清楚。因此，必须使用约束性更强的方式表达过程细节。

对于详细设计工具，要能提供对设计的无歧义性描述，能指明控制流程、处理功能、数据组织等实现细节，从而在编程阶段能把对设计的描述直接翻译成程序代码。

6.3.1 流程图

流程图(flow chart)即程序框图，又称程序流程图，是用统一规定的标准符号描述程序执行具体步骤的图形表示，是使用历史最久、流行最广的一种描述工具。从 20 世纪 40 年代末到 70 年代中期，流程图一直是软件设计的主要工具。

1. 基本成分

流程图包括三种基本成分：

(1) 处理，用方框表示。

（2）判断条件，用菱形框表示。

（3）控制流，用箭头表示。

2. 基本的控制结构

为使用流程图描述结构化程序，必须限定在流程图中使用的控制结构。通常只允许五种基本的控制结构。

（1）顺序型：由几个连续的加工步骤依次排列构成。

（2）选择型：由某个逻辑判断式的取值决定选择两个加工中的一个。

（3）先判断（While）型循环：在循环控制条件成立时，重复执行特定的加工。

（4）后判断（Until）型循环：重复执行某些特定的加工，直到控制条件成立。

（5）多情况（Case）选择型：列举了多种加工情况，根据控制变量的取值，选择执行其一。

前四种控制结构如图 6.1 所示，多情况选择型的图形化描述如图 6.2 所示。

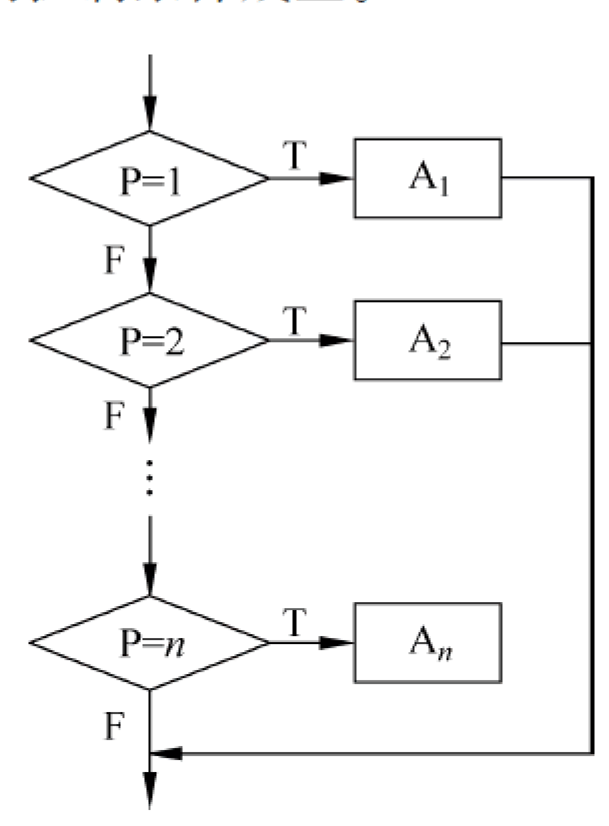


图 6.2　多情况选择型

3. 优点

对控制流程的描绘直观、清晰，易于学习掌握。

4. 缺点

（1）本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

（2）用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的思想，随意转移控制。

（3）不易表示数据结构。

（4）修改麻烦。

6.3.2　盒图

1983 年，美国的 I. Nassi 和 B. Sheiderman 共同提出了一种不用 Goto 语句、不需要流向线的结构化流程图，称为盒图，也称为 N-S 图。

在 N-S 图中，每个处理步骤用一个盒子表示，盒子可以嵌套。盒子只能从上头进入，从下头走出，除此之外别无其他出入口，所以盒图限制了随意的控制转移，保证了程序的良好结构。

1. 基本控制结构

五种基本控制结构表示如图 6.3(a)～(e)所示。盒图也可以表示调用子程序，如图 6.3(f)所示。

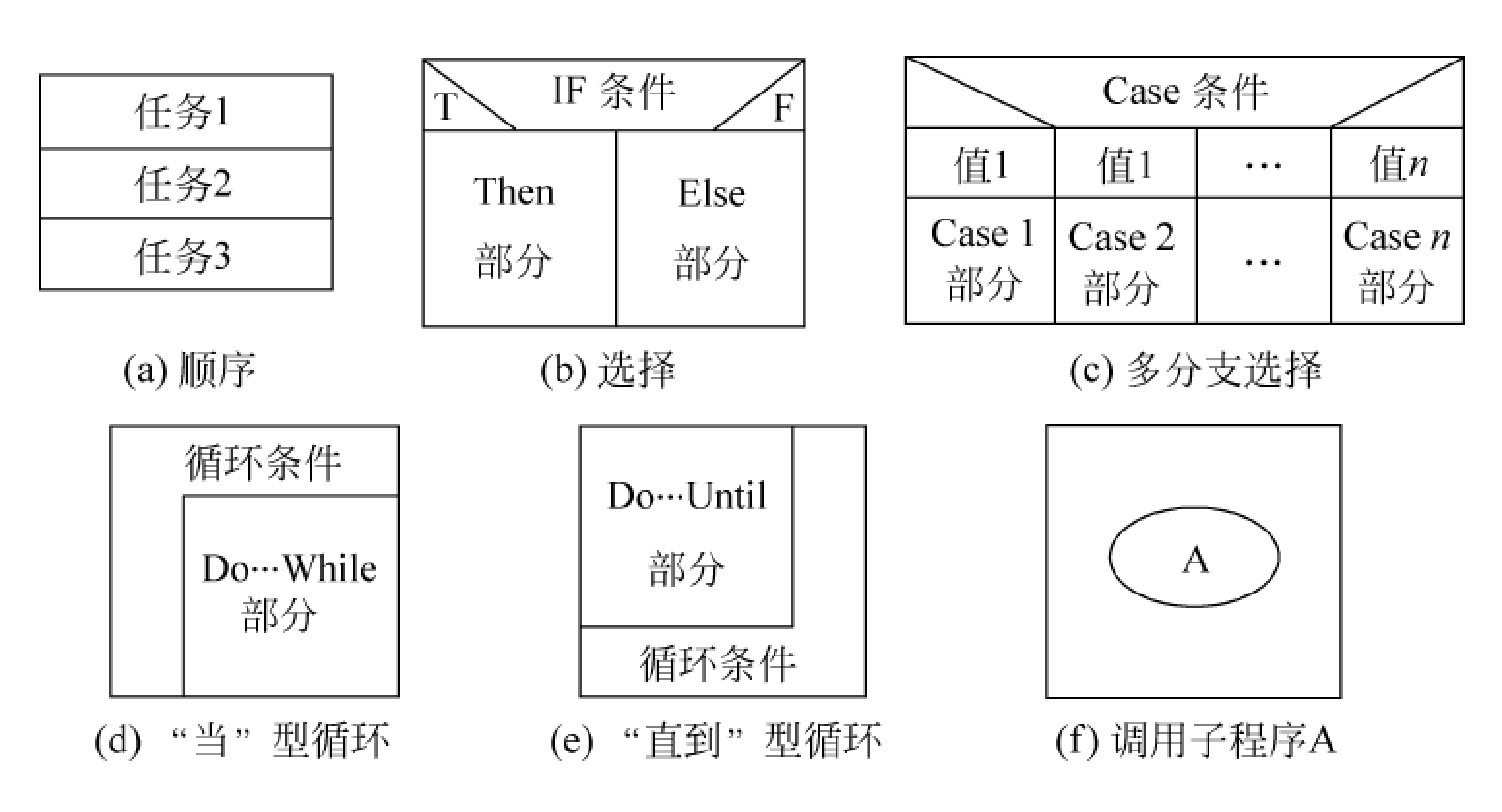


图 6.3　盒图的基本控制结构

2. 优点

(1) 功能域(即一个特定控制结构的作用域)明确,可以从盒图上一眼就看出来。

(2) 没有箭头,无法随意转移控制。

(3) 很容易确定局部和全程数据的作用域。

(4) 很容易表示嵌套关系,也可以表示模块的层次结构。

(5) 强制设计人员按结构程序设计方法进行思考和描述其方案,由 N-S 图得到的程序必定是结构化的。

(6) 图形直观,容易理解设计意图,为编程、复查、测试、维护带来方便。

3. 缺点

(1) 当分支嵌套层次较多时,有时在一页纸上很难画下。

(2) 画图麻烦,修改麻烦。

6.3.3　问题分析图

问题分析图(Problem Analysis Diagram,PAD)是由日本日立公司二村良彦等人于 1979 年提出的,是一种支持结构程序设计的图形工具。问题分析图通过用二维树形结构的图形来表示程序的控制流,将这种图形翻译成程序代码比较容易,已经得到一定程度的推广。

PAD 图既克服了传统的流程图不能清晰表达程序结构的缺点,又不像 N-S 图那样受到把全部程序约束在一个方框内的限制,不仅逻辑结构清晰、图形标准,而且更重要的是能引导人们使用结构化的程序设计方法,从而有利于提高程序设计的质量。以 PAD 图为基础,按照机械的变换规则,就可以写出结构化的程序。

1. 基本控制结构表示

五种基本控制结构表示如图 6.4(a)~(e)所示。

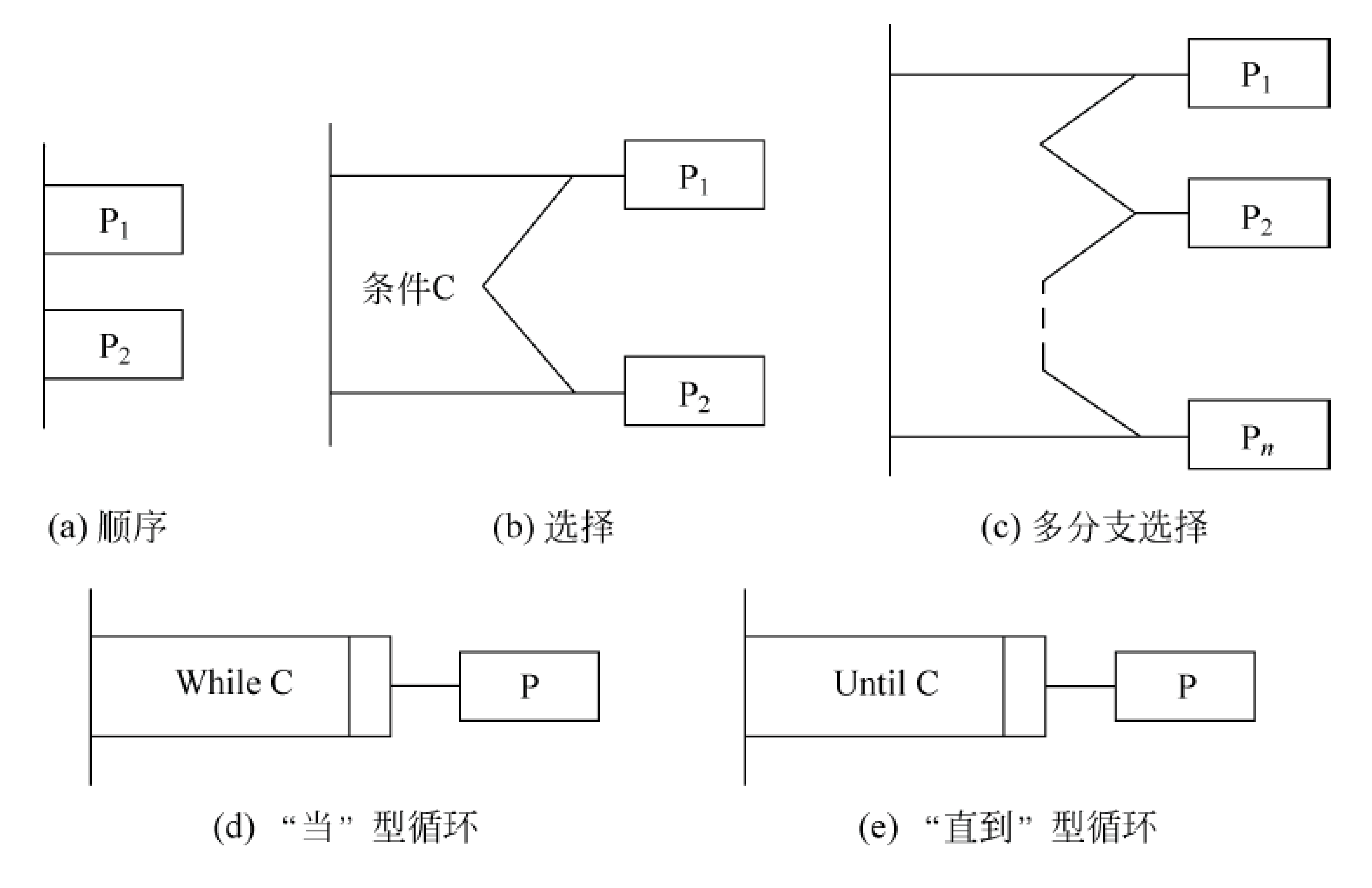


图 6.4 问题分析图的基本控制结构

2. 优点

(1) 使用 PAD 符号所设计出来的程序必然是结构化程序。

(2) PAD 图描绘的程序结构十分清晰。图中最左边的竖线是程序的主线,即第一层结构。随着程序层次的增加,PAD 图逐渐向右延伸,每增加一个层次,图形向右扩展一条竖线。PAD 图中竖线的总条数就是程序的层次数。

(3) 用 PAD 图表示程序逻辑,易读、易懂、易记。

(4) PAD 图是二维树形结构的图形,程序从图中最左竖线上端的结点开始执行,自上而下、从左向右顺序执行,遍历所有结点。

(5) 容易将 PAD 图转换成高级语言源程序,这种转换可用软件工具自动完成,从而省去人工编码工作,有利于提高软件可靠性和软件生产率。

(6) 可用于表示程序逻辑,也可用于描绘数据结构。

(7) PAD 图的符号支持自顶向下、逐步求精方法的使用。开始时设计者可以定义一个抽象的程序,随着设计工作的深入,而使用 def 符号逐步增加细节,直至完成详细设计。

6.3.4 过程设计语言

过程设计语言(Procedure Design Language,PDL)是用来描述模块内部具体算法的非正式且比较灵活的语言,外层语法是确定的,而内层语法不确定。外层语法描述控制结构,用类似一般编程语言的保留字,所以是确定的。内层语法不确定,可以根据系统的具体情况和不同层次灵活选用,实际上可以采用任意自然语言来描述具体操作。由于 PDL 与程序很相似,所以也称为伪程序或伪代码。但它仅仅是对算法的一种描述,是不可执行的。

1. 基本控制结构

基本控制结构如图 6.5 所示。在原来五种基本控制结构的基础上加入了“For”型循环结构。

```
(a) 顺序
    处理 S1
    处理 S2
     …
    处理 Sn
(b) 选择
① If 结构：
    If 条件
      处理 S1
    Else
      处理 S2
    EndIf
或：
    If 条件
      处理 S
    EndIf
```

```
② If…OrIf…Else 结构：
    If 条件 1
      处理 S1
      OrIf  条件 2
       …
    Else 处理 Sn
    EndIf
(c) 多分支选择
    Case Of
      Case(1)
        处理 S1
      Case(2)
        处理 S2
       …
      Else 处理 Sn
    EndCase
```

```
(d) “当”型循环
    While 条件
       循环体
    EndWhile
(e) “直到”型循环
    Repeat
       循环体
    Until 条件
(f) “For”型循环
    For i=1 To n
       循环体
    End For
```

图 6.5 过程设计语言的基本控制结构

2. 优点

PDL 的总体结构与一般程序完全相同。外层语法同相应的程序语言一致，内层语法使用自然语言，易编写、易理解，也很容易转换成源程序。除此以外，还有以下优点：

(1) 提供的机制较图形全面，为保证详细设计与编码的质量创造了有利条件。

(2) 可作为注释嵌入在源程序中一起构成程序文档，并可同高级程序设计语言一样，进行编辑、修改，有利于软件维护。

(3) 可自动生成程序代码，提高软件生产率。目前 PDL 已有多种版本(如 PDL/PASCAL、PDL/C、PDL/Ada 等)，为自动生成相应代码提供了便利条件。

3. 缺点

(1) 不如图形工具形象直观，对英语使用的准确性要求较高。

(2) 描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

6.3.5 IPO 图

IPO(Input Process Output)图是用于描述某个特定模块内部的处理过程和输入输出关系的图形。IPO 是配合 HIPO 详细说明每个模块的输入数据、输出数据和数据加工的重要工具。常用的 IPO 图的基本内容如表 6.1 所示。

表 6.1 IPO 图的基本内容

系统名称：	
模块名称：	模块编号：
模块描述：	
被调用模块：	调用模块：
输入参数：	输入说明：
输出参数：	输出说明：
变量说明：	
使用的文件或数据库：	
处理说明：	
备注：	
设计人：	设计日期：

IPO 图的主体是处理说明部分，该部分可采用流程图、N-S 图、问题分析图和过程设计语言等工具进行描述，几种方法各有长处和不同的适用范围，在实际工作中究竟采用哪一种，需视具体情况和设计者的习惯而定，选用的基本原则是能准确而简明地描述模块执行的细节。

在 IPO 图中，输入数据、输出数据来源于数据字典。变量说明是指模块内部定义的变量，与系统的其他部分无关，仅由本模块定义、存储和使用。备注是对本模块有关问题做必要的说明。开发人员不仅可以利用 IPO 图进行模块设计，而且还可以利用它评价总体设计。用户和管理人员可利用 IPO 图编写、修改和维护程序。因此，IPO 图是详细设计阶段的一种重要文档资料。

6.3.6 判定表

当算法中包含多重嵌套的条件选择时，用流程图、盒图、PAD 图或过程设计语言（PDL）等详细设计工具都不易清楚地描述，但判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

一张判定表由四部分组成：左上部列出所有条件；左下部是所有可能的动作；右上部是表示各种条件组合的一个矩阵；右下部是和每种条件组合相对应的动作。判定表右半部的每一列实质上是一条规则，规定了与特定的条件组合相对应的操作。

某高校分房的判定表如表 6.2 所示。

表 6.2 某高校分房的判定表

工作年限		1	2	3	4	5	6	说　明
条件	婚龄	＞5 年		3～5 年			＜3 年	
	中级职称	---		Y		N	---	(---)：任意
	晚婚	Y	N	Y	N	---	---	(Y)：条件满足
决策	优先分房	☆		☆				(N)：条件不满足
	正常分房		☆		☆			(☆)：选中的决策
	不分房					☆	☆	

从高校分房的例子可以看出,判定表能够简洁而又无歧义地描述处理规则。当把判定表和布尔代数或卡诺图结合起来使用时,可以对判定表进行校验或化简。但是判定表并不适合作为一种通用的设计工具,没有一种简单的方法使它能同时清晰地表示顺序和重复等处理结构。

6.3.7 判定树

判定表虽然能清楚地表达复杂的条件组合与应做的动作之间的对应关系,但当条件组合较多时不够清晰,简洁程度下降。

判定树是判定表的变种,也能清晰地表示复杂的条件组合与应做的动作之间的对应关系。优点在于,形式简单到不需任何说明,一眼就可以看出含义,因此易于掌握和使用。多年来判定树一直受到人们的重视,是一种比较常用的系统分析和设计工具。

某高校分房的判定树如图 6.6 所示。

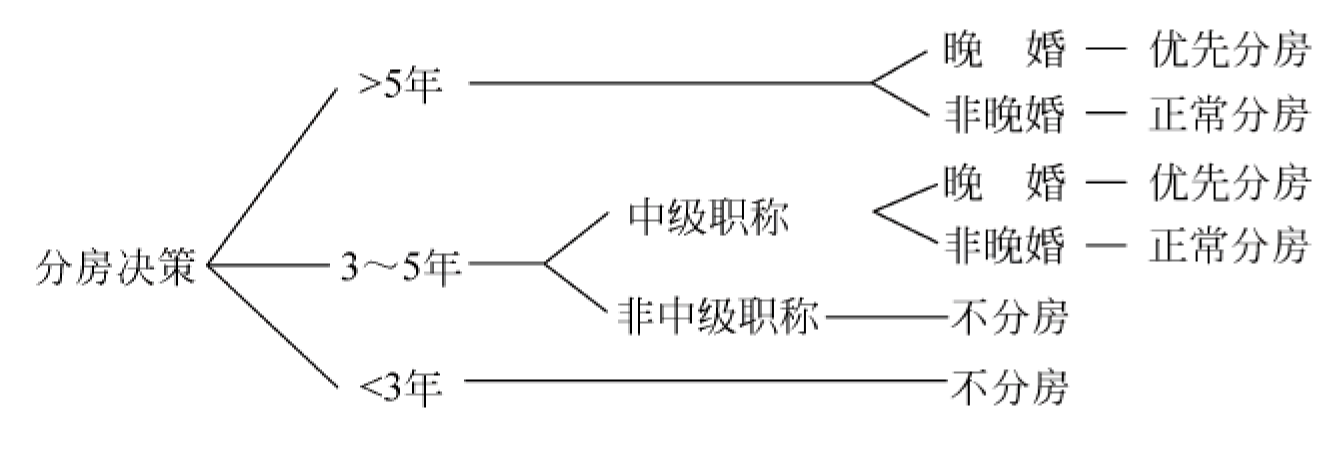


图 6.6 某高校分房的判定树

从高校分房的例子可以看出,虽然判定树比判定表更直观,但简洁性不如判定表,数据元素的同一个值往往要重复写多遍,而且越接近树的叶端重复次数越多。另外,画判定树时分支的次序可能对最终画出的判定树的简洁程度有较大影响,而判定表并不存在这样的问题,所以要画好判定树需要一定的技巧。

6.4 面向数据结构的设计方法

在许多应用领域信息都有清楚的层次结构,输入数据、内部存储信息以及输出数据都有独特的结构。数据结构既影响程序结构又影响程序处理过程,重复出现的数据用循环控制结构的程序处理,选择数据用分支控制结构的程序处理。

面向数据结构的设计方法,就是用数据结构作为程序设计的基础,最终目标是得出对程序处理过程的描述,最适合于在详细设计阶段使用。在完成了软件结构设计之后,可以使用面向数据结构的方法来设计每个模块的处理过程。使用面向数据结构的设计方法首先需要分析确定数据结构,并且用适当的工具清晰地描述数据结构。

6.4.1 Jackson 系统开发方法

Jackson 系统开发(Jackson System Development,JSD)方法是一种典型的面向数据结构的分析设计方法。

1. Jackson 图表达基本结构

对于种类繁多的程序中使用的数据结构，各数据元素之间的逻辑关系只有顺序、选择、重复三种，所以逻辑数据结构也只有三种，如图 6.7 所示。

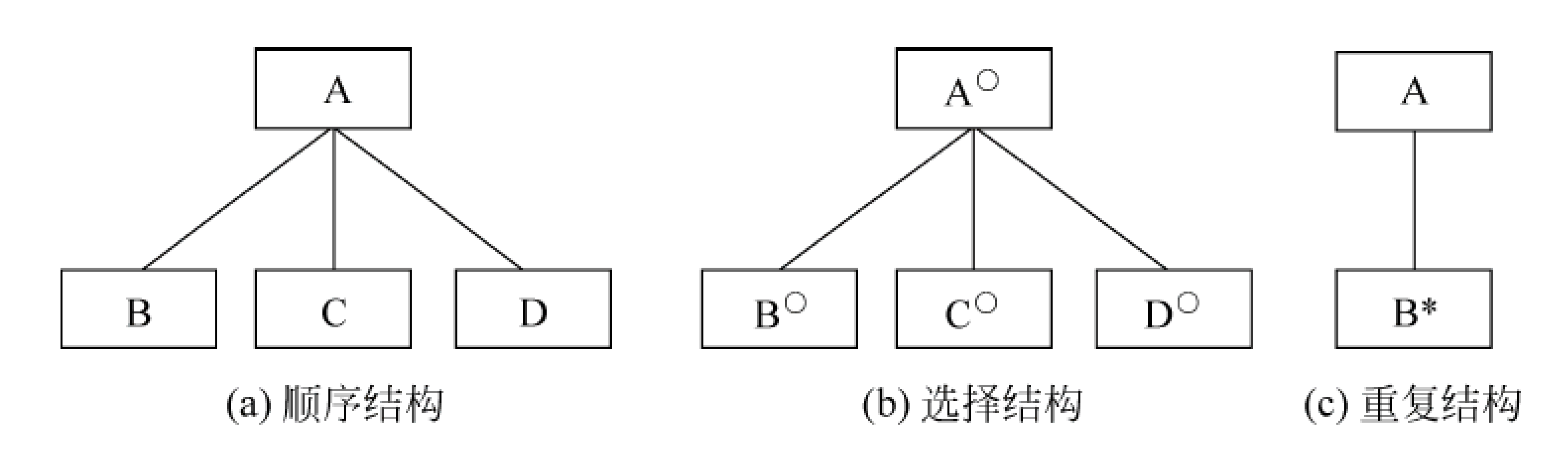


图 6.7 Jackson 图的三种结构

(1) 顺序结构。数据由一个或多个数据元素组成，每个元素按确定次序出现一次。图 6.7(a)中，A 由 B、C、D 三个元素顺序组成，出现的次序为 B、C、D。

(2) 选择结构。包含两个或多个数据元素，每次使用这个数据时按一定条件从这些数据元素中选择一个。图 6.7(b)中，条件 A 是 B、C、D 中的某一个。B、C、D 的右上角有小圆圈做标记。

(3) 重复结构。根据使用时的条件，由一个数据元素出现零次或多次组成。图 6.7(c)中，A 由 B 出现 N 次($N\geqslant 0$)组成或 A 由 B 循环组成，结束条件的编号是 i。B 的右上角有星号标记。

2. 改进的 Jackson 图

Jackson 图的缺点是表示选择或重复结构时，选择条件或循环结束条件不能直接在图上表示出来，影响了图的表达能力，也不易直接把图翻译成程序，此外，框间连线为斜线，不易在行式打印机上输出。建议用改进的 Jackson 图，如图 6.8 所示。

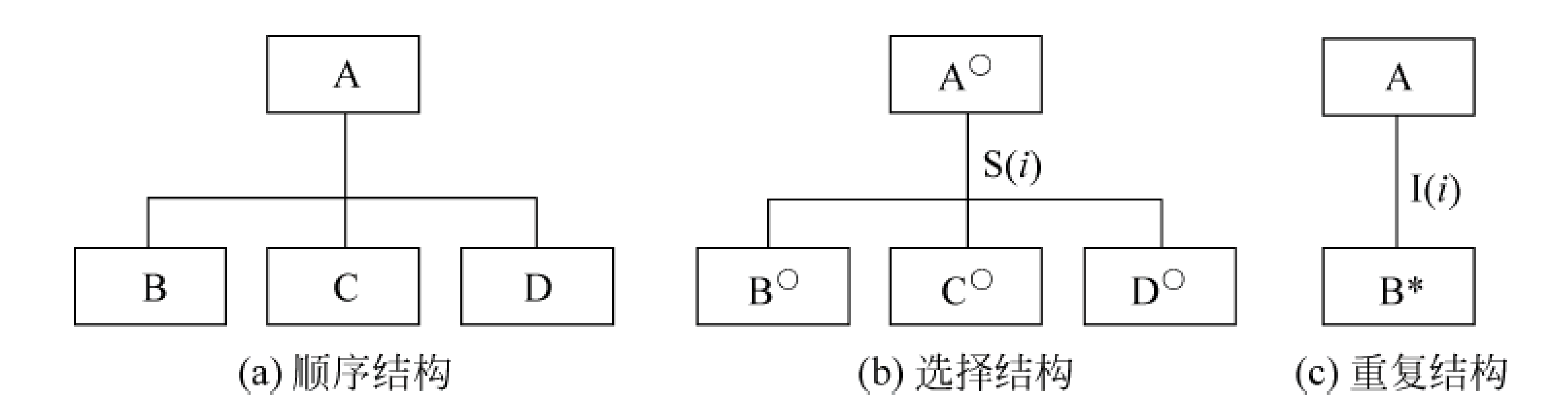


图 6.8 改进的 Jackson 图

(1) 顺序结构。图 6.8(a)中，B、C、D 中任何一个都不能是选择出现或重复出现的数据元素，即不能是右上角有小圆圈或星号标记的元素。

(2) 选择结构。图 6.8(b)中，S 右边括号内的 i 是分支条件的编号。

(3) 重复结构。图 6.8(c)中，循环结束条件的编号为 i。

3. 用 Jackson 图表示数据结构

用 Jackson 图表示二维表格，如图 6.9 所示。

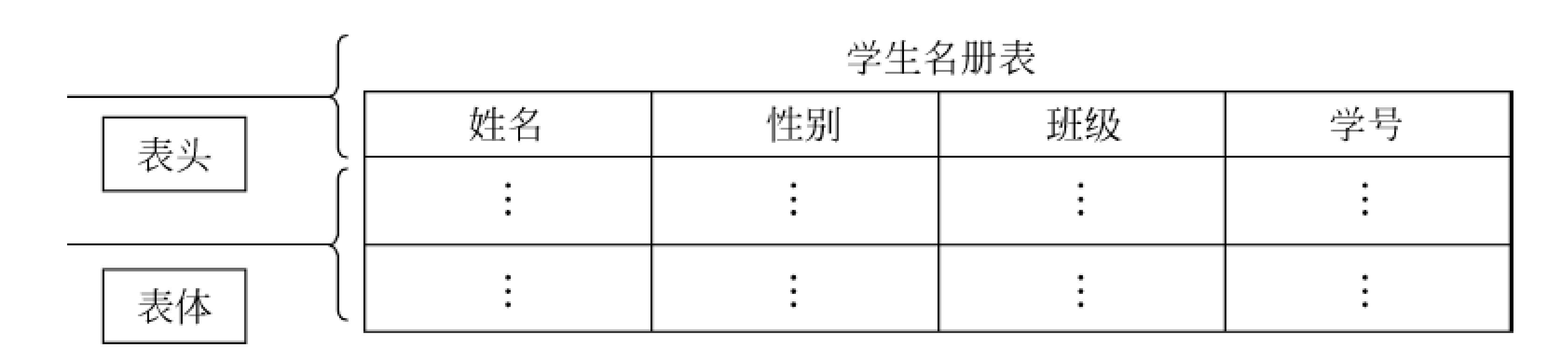


图 6.9 学生名册表

如图 6.9 所示的 Jackson 图首先声明了学生名册表格由表头和表体两部分组成。其中,表头又包括表名和字段名,而表体可由任意行(0 行或多行)组成,每行包括学生的姓名、性别、班级和学号。班级是本科的,学号项是本科生学号;班级是研究生的,学号项是研究生学号。

4. 用 Jackson 图表示程序结构

用 Jackson 图表示产生上面学生名册文件程序的程序结构,把学生名册生成为一个计算机文件,则该程序结构的 Jackson 图可以如图 6.10 所示。

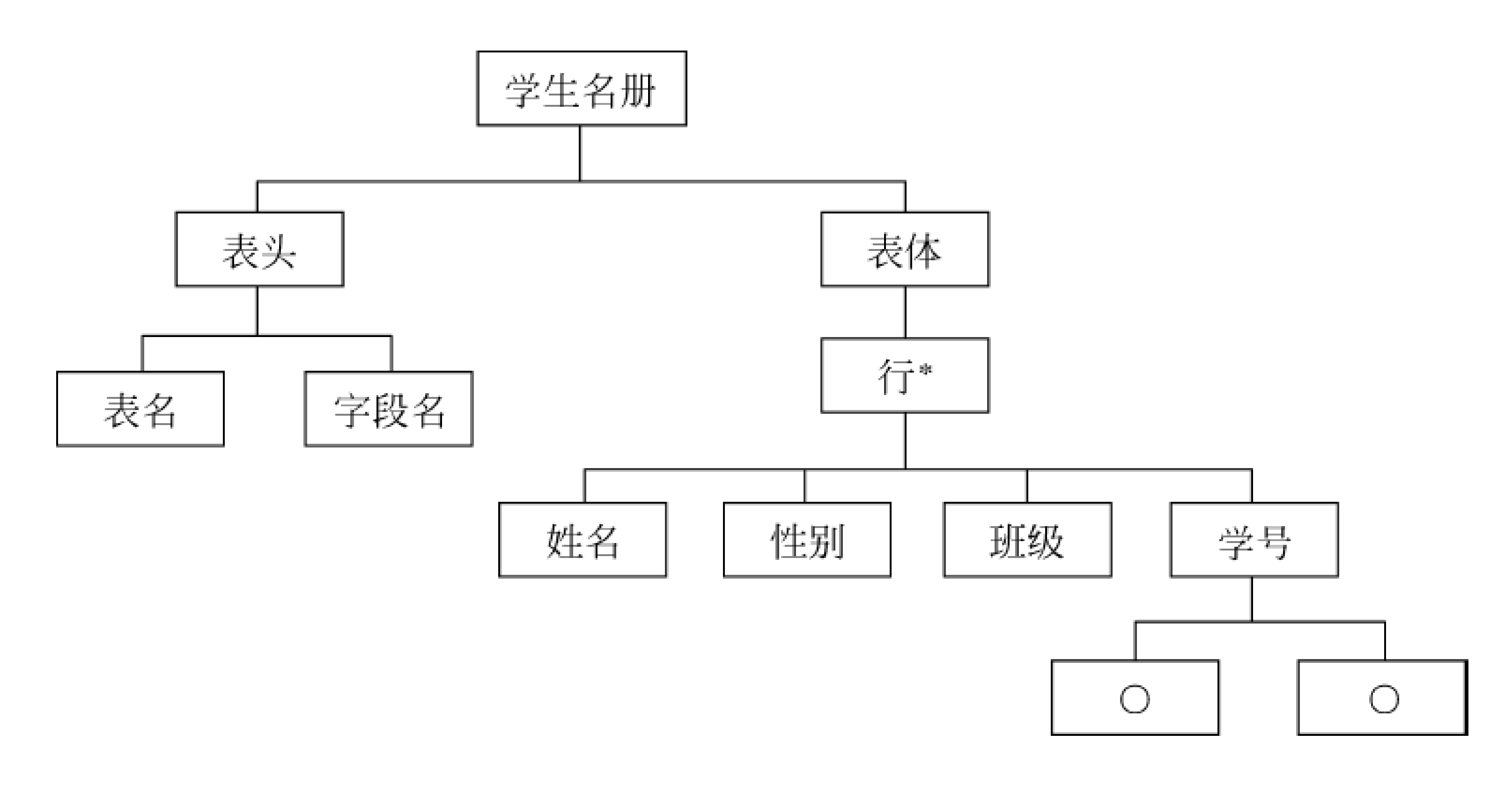


图 6.10 用 Jackson 图表示学生名册

5. Jackson 伪代码

Jackson 系统开发方法中使用的伪代码与 Jackson 图是完全对应的,三种基本结构对应的伪代码如图 6.11 所示。

(a) 顺序结构	(b)选择结构	(c) 重复结构
A seq do B; do C; do D; A end	A select cond1 do B; or cond2 do C; or cond3 do D; A end	A iter until(或 while)cond do B; A end

图 6.11 Jackson 基本结构的伪代码

(1) 顺序结构。图 6.11(a)中，seq 和 end 是关键字。

(2) 选择结构。图 6.11(b)中，select、or、do 和 end 是关键字，cond1、cond2 和 cond3 分别是执行 B、C 或 D 的条件。

(3) 重复结构。图 6.11(c)中，iter、until、while、do 和 end 是关键字(重复结构有 until 和 while 两种形式)，cond 是条件。

6. Jackson 系统开发方法步骤与实现

(1) 实体动作分析。

(2) 实体结构分析。

(3) 定义初始模型。

(4) 确定输入数据和输出数据的逻辑结构，并用 Jackson 图描绘这些数据结构。

(5) 找出输入数据结构和输出数据结构中有对应关系的数据单元。

(6) 列出完成结构图各框处理功能的全部操作和条件，并且把它们分配到程序结构图的适当位置。

(7) 用伪码表示程序。

6.4.2 Warnier 方法

Warnier 方法是另一种面向数据结构的设计方法，又称为逻辑构造程序的方法，简称 LCP(Logical Construction of Programs)方法。Warnier 方法的原理和 Jackson 系统开发方法类似，也是从数据结构出发设计程序，只是这种方法的逻辑更严格。Warnier 图是在 Warnier 方法中使用的一种专用表达工具。

1. Warnier 图

Warnier 图是由嵌套的大括号、伪代码以及少量说明和符号组成的层次树。Warnier 图也具有表达数据结构和程序结构的双重功能。

2. Warnier 设计方法

(1) 分析和确定输入数据和输出数据的逻辑结构，并用 Warnier 图描绘这些数据结构。

(2) 主要依据输入数据结构导出程序结构，并用 Warnier 图描绘程序的处理层次。

(3) 画出程序流程图并自上而下地给每个处理框编序号。

(4) 分类写出伪码指令。

(5) 把前一步中分类写出的指令按序号排序，从而得出描述处理过程的伪码。

6.5 程序复杂性度量

程序复杂性主要指模块内程序的复杂性，直接关系到软件开发费用的多少、开发周期的长短和软件内部潜伏错误的多少。

程序复杂性度量的意义在于，程序的复杂程度乘以适当常数即可估算出软件中故障的

数量以及软件开发需要的工作量；可以比较两个不同设计和算法的优劣；定量的复杂程度可作为模块规模的精确限制。

为了度量程序复杂性，要求复杂性度量满足以下假设：可以用来计算任何一个程序的复杂性；对于不合理的程序，例如长度动态增长的程序，或者原则上无法排错的程序，不应当使用它进行复杂性计算；如果程序中指令条数、附加存储量、计算时间等增多，不会减少程序的复杂性。

程序复杂性度量的方法很多，在此介绍以下几种。

1. 代码行度量法

程序复杂性度量最简单的方法就是统计程序的源代码行数。此方法的基本思想是统计程序的源代码行数，并以源代码行数作为程序复杂性的度量。

代码行度量法基于两个前提：

（1）程序复杂性随着程序规模的增加不均衡地增长。

（2）控制程序规模的方法最好是分而治之，即将一个大程序分解成若干个简单的可理解的程序段。

设每行代码的出错率为每100行源程序中可能有的错误数目，例如每行代码的出错率为1%，则是指每100行源程序中可能有一个错误。Thayer指出，程序出错率的估算范围为0.04%～7%，且每行代码的出错率与源程序行数之间不存在简单的线性关系。Lipow进一步指出，对于小程序，每行代码的出错率为1.3%～1.8%；对于大程序，每行代码的出错率增加到2.7%～3.2%之间，但这只是考虑了程序的可执行部分，没有包括程序中的说明部分。Lipow及其他研究者得出这样的结论："对于少于100个语句的小程序，源代码行数与出错率是线性相关的；随着程序的增大，出错率以非线性方式增长。"所以，代码行度量法只是一个简单的、粗糙的方法。

2. McCabe 度量法

McCabe（麦坎比）度量法是一种基于程序控制流的复杂性度量方法。McCabe定义的程序复杂性度量值又称为环路复杂度，基于一个程序模块的程序图中环路的个数。

如果把程序流程图中每个处理符号都退化成一个结点，原来连接不同处理符号的流线变成连接不同结点的有向弧，这样得到的有向图就叫做程序图。

计算有向图G的环路复杂性公式为：

$$V(G) = m - n + 2 \tag{6.1}$$

其中，$V(G)$是有向图G中的环路个数，m是图G中的有向弧个数，n是图G中的结点个数。以图6.12为例，弧数$m=12$，结点数$n=11$，则有：

$$V(G) = m - n + 2 = 12 - 11 + 2 = 3$$

即McCabe环路复杂度量值为3。它也可以看做由程序图中的有向弧所封闭的区域个数。

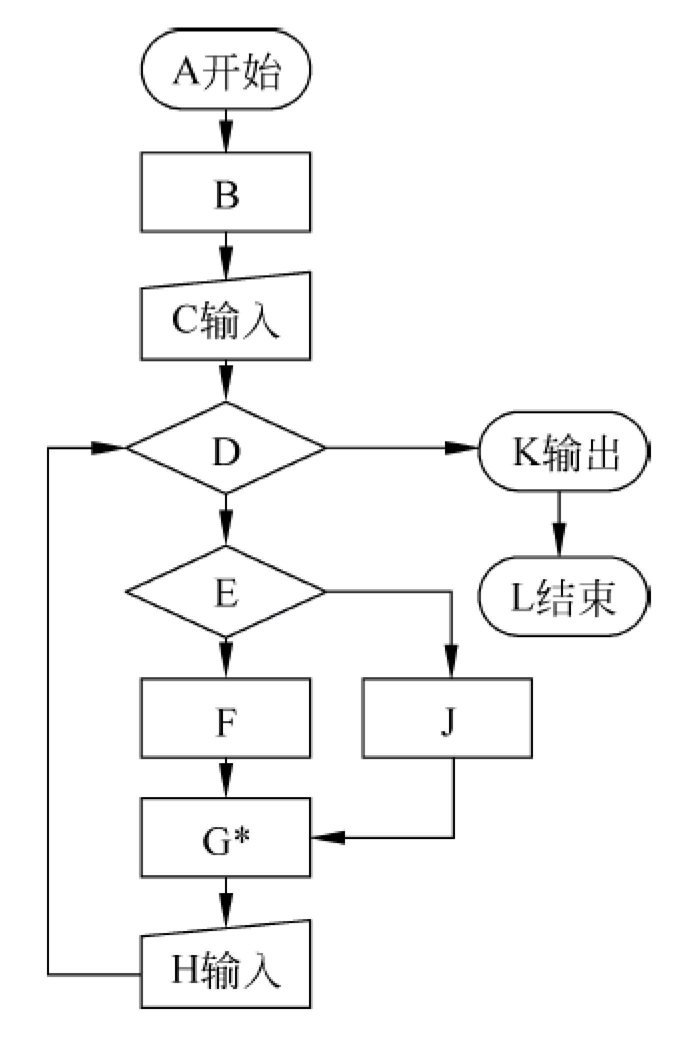


图6.12 程序图示例

当分支或循环的数目增加时，程序中的环路也随之增加，因此 McCabe 环路复杂度量值实际上是为软件测试的难易程度提供了一个定量度量的方法，同时也间接地表示了软件的可靠性。实验表明，源程序中存在的错误数以及为了诊断和纠正这些错误所需的时间与 McCabe 环路复杂度量值有明显的关系。

Myers 建议，对于复合判定，例如$(A=0)\cap(C=D)\cup(X='A')$，应算做三个判定。

利用 McCabe 环路复杂度量时，有以下几点说明：

(1) 环路复杂度取决于程序控制结构的复杂度。当程序的分支数目或循环数目增加时复杂度也增加。环路复杂度与程序中覆盖的路径条数有关。

(2) 环路复杂度是可加的。例如，模块 A 的复杂度为 3，模块 B 的复杂度为 4，则模块 A 与模块 B 的复杂度是 7。

(3) McCabe 建议，对于复杂度超过 10 的程序，应分成几个小程序，以减少程序中的错误。Walsh 用实例证实了这个建议的正确性。他发现，在 McCabe 复杂度为 10 附近时，存在出错率的间断跃变。

(4) McCabe 环路复杂度隐含的前提是：错误与程序的判定加上例行子程序的调用数目成正比，而加工复杂性、数据结构、录入与打乱输入卡片的错误可以忽略不计。

3. Halstead 度量法

Halstead 方法根据程序中运算符和操作数的总数来度量程序的复杂度。程序长度 N 定义为：

$$N = N_1 + N_2 \tag{6.2}$$

其中，N_1 表示运算符总数，N_2 表示操作数总数。

若已知程序中使用的不同运算符个数 n_1 和不同操作数个数 n_2，则预测程序长度的公式为：

$$H = n_1\log_2 n_1 + n_2\log_2 n_2 \tag{6.3}$$

预测程序中错误个数的公式为：

$$E = N\log_2(n_1 + n_2)/3000 \tag{6.4}$$

例如，一个程序对 75 个数据库项共访问 1300 次，对 150 个运算符共使用 1200 次，那么预测该程序的错误数为：

$$E = (1300 + 1200) * \log_2(75 + 150)/3000 = 6.5$$

即预测该程序中可能含有 6～7 个错误。

Halstead 的重要结论：程序的实际 Halstead 长度 N 可以由词汇表 n 算出。即使程序还未编制完成，也能预先算出程序的实际 Halstead 长度 N，虽然没有明确指出程序中到底有多少个语句。

这个结论非常有用。经过多次验证，预测的 Halstead 长度与实际的 Halstead 长度非常接近。

思考题

1. 详细设计的任务是什么？
2. 结构程序设计有哪些基本控制结构？如何表示？

3. 结构程序设计技术的优越性体现在哪些方面?

4. 了解流程图、盒图、问题分析图、过程设计语言、IPO 图、判定表、判定树等详细设计工具,掌握你认为最常用的三种工具的使用方法。

5. 流程图的优缺点分别是什么?

6. 盒图的优缺点分别是什么?

7. 问题分析图的优点有哪些?

8. 过程设计语言的优缺点分别是什么?

9. Jackson 图如何表示数据结构和程序结构?

10. 了解 Warnier 设计方法。

11. 程序复杂性度量有哪些方法?

12. 理解 McCabe 度量法的计算有向图 G 的环路复杂性公式。

第7章 软件实现

软件实现是软件工程过程中的重要阶段，其任务是根据软件设计的结果，编写正确的、易于理解和维护的程序模块，并对这些程序模块进行调试和单元测试。编程也称程序设计，是指用程序设计语言编写计算机程序，是软件实现的关键，通常采用结构化编程和面向对象编程等方法。

单元测试内容放置到第 8 章软件测试进行讲述。本章讲述软件实现相关的输入输出设计、屏幕界面设计、编程语言、编程风格等内容。

7.1 输入设计

输入输出(Input/Output，I/O)设计是在设计过程中很容易被忽视的环节，又是一个重要环节，对用户使用的方便性、安全性和可靠性都非常重要。一个好的输入系统设计可以为用户带来良好的工作环境；一个好的输出设计可以为管理者提供简捷、明了、有效、实用的管理和控制信息。本节讲述输入设计，7.2 节讲述输出设计。

输入设计对系统质量有着决定性的影响。输入数据的正确性直接决定处理结果的正确性，如果输入数据有误，即使计算和处理十分正确，也无法获得可靠的输出信息。同时，输入设计是软件系统与用户之间交互的纽带，决定着人机交互的效率。在软件开发的实现过程中输入设计所占的比重较大，一个好的输入设计能为今后系统运行带来很多方便。

7.1.1 设计原则

输入设计的目标是在保证向软件系统提供正确信息和满足需要的前提下，尽可能做到输入方法简单、迅速、经济和方便使用者。输入设计必须根据输出设计的要求来确定，并遵循以下原则：

(1) 控制输入量。输入量应保持在能满足处理要求的最低限度，避免不必要的重复与冗余。输入量越少，错误率越小，数据准备时间也越少。

(2) 减少输入延迟。输入数据的速度往往成为提高软件系统运行效率的瓶颈，为减少延迟，可采用中转文件、批量输入等方式。

(3) 减少输入错误。输入准备及输入过程应尽量简易、方便，并有适当查错、防错、纠错措施，从而减少错误发生。

(4) 避免额外步骤。在输入设计时，应尽量避免不必要的输入步骤，当某步骤不能省略

时,应仔细验证现有步骤是否完备、高效。

(5) 尽早保存。输入数据时尽量用处理所需的形式记录下来,以避免数据由一种介质转换到另一种介质时,需要转录及可能发生错误。

(6) 及时检查。应尽早对输入数据进行检查,以便使错误及时得到改正。

7.1.2 输入方式

输入方式设计主要是根据详细设计和数据库设计的要求,来确定数据输入的具体形式。常用的输入方式有键盘输入、模/数输入、数/模输入、网络数据传送、磁盘/光盘读入等几种。通常设计新系统的输入方式时,尽量利用已有的设备和资源,避免大批量数据的重复键盘输入。因为键盘输入不但工作量大、速度慢,而且出错率高。

(1) 键盘输入(Key-In)。键盘输入方式包括联机键盘输入和脱机键盘输入(通过键到磁盘、键到磁带等设备,将数据输入到磁盘、磁带文件中,然后再读入到系统设备)两种方式。主要适用于常规、少量的数据和控制信息的输入,以及原始数据的输入。这种方式不太适合大批中间处理数据的输入。

(2) 数模/模数转换方式(A/D、D/A)。数模/模数转换方式输入是目前比较流行的基础数据输入方式,是直接通过光电设备对实际数据进行采集,并将其转换成数字信息的方法,是一种既省事,又安全可靠的数据输入方式。常见的有如下几种方法:

① 条码(棒码)输入。这种方式利用标准的商品分类和统一规范化的条码,贴(或印)于商品的包装上,通过光学符号阅读器(Optical Character Reader,OCR,亦称扫描仪)来采集和统计商品的流通信息。这种数据采集和输入方式现已普遍应用于商业、企业、工商、质检、海关、图书馆等软件系统中。

② 扫描仪输入。这种方式实际上与条码输入是同一类型,被大量地使用在图形/图像的输入、文件/报纸的输入、标准考试试卷的自动阅卷、投票的统计等应用中。

③ 传感器输入。这种方式利用各类传感器和电子衡器,接收和采集物理信息,然后再通过 A/D 板将其转换为数字信息。这也是一种用于采集和输入生产过程数据的方法。

(3) 网络传送数据。这既是一种输出信息方式,又是一种输入信息方式。对下级子系统是输出,对上级主系统是输入。使用网络传送数据既安全、可靠、快捷,又可避免下级忙于设计输出界面、上级忙于设计输入界面的重复性开发工作。网络传送有两种方式:

① 利用数字网络直接传送数据。

② 利用电话网络(modem)传送数据。

(4) 磁盘传送数据。磁盘传送数据是数据输出和接收双方事先约定好待传送数据文件的标准格式,然后通过软盘/光盘传送数据文件。这种方式不需要增加任何设备和投入,是一种非常方便的输入数据方式,目前还常被用在主/子系统之间的数据联结上。

7.1.3 输入格式

设计数据输入格式时,应严格按照数据库设计时产生的数据字典,遵循代码设计的实际标准,统一格式。

但在一些旧系统改造过程中,实际数据输入时(特别是大批量的数据统计报表输入),有

时会遇到统计报表(或文件)结构与数据库文件结构不完全一致的情况。这时应尽量严格参照有关标准,统一格式,不能随意更改数据库结构。在特殊情况下,应专门编制一个转换模块,以适应特殊要求。

现在还可以采用智能输入方式,由计算机自动将输入数据送至不同表格中。

输入设计的重要内容之一是设计好原始单据格式。研制新系统时,即使原系统的单据很齐全,一般也要重新设计和审查原始单据。

设计原始单据的原则如下:

(1) 便于填写。原始单据的设计要保证填写迅速、正确、全面、简易和节约,具体地说应做到填写量小、版面排列简明、易懂。

(2) 便于归档。单据大小要标准化,预留装订位置,标明传输的流动路径。

(3) 单据的格式应能保证输入精度。

7.1.4 输入校验

输入设计要尽可能减少数据输入的错误,在输入设计时,要对全部输入数据设想可能发生的错误,对其进行校验。

1. 输入错误的种类

输入错误通常有以下几种:

(1) 数据本身错误。由于原始数据填写错误引起的输入数据错误。

(2) 数据多余或不足。这是在数据收集过程中产生的差错。如数据(单据、卡片等)的丢失、遗漏或重复等原因引起的错误。

(3) 数据延误。数据延误也是数据收集过程中产生的差错,不过内容和数据都是正确的,只是由于时间上的延误而产生差错。这种差错多由开票、传送等环节的延误而引起,严重时,会导致输出信息无利用价值。因此,数据的收集与运行必须具有一定的时间性,并要事先确定产生数据延迟时的处理对策。

2. 校验方法

数据校验方法有人工直接检查、计算机程序校验、人与计算机分别处理后再相互查对校验等多种方法。常用的方法有以下几种,可单独使用,也可组合使用。

(1) 静态校验:人工校验。这种方法一般是在输入之前,由人工对数据进行检查。也可在数据输入之后,由计算机将输入的有关数据重新输出,然后由人工将输出的数据与原始数据逐个核对,检查是否一致。这种方法适用于少量的数据或控制字符输入,但对大批量的数据输入过于麻烦,效率太低。

(2) 二次输入校验:指同一批数据两次输入系统的方法。输入后系统内部再比较这两批数据,如果完全一致则认为输入正确;反之,则将不同部分显示出来,有针对性地由人工进行校对。该方法的好处是方便、快捷,而且可以用于任何类型的数据符号。尽管该方法中二次输入会在同一地方出错,并且错误一致的可能性是存在的,但是这种可能性出现的概率极小,是目前常用的方法。

(3) 逻辑校验:对输入的数据是否符合逻辑性,有关数据的值是否合理的一种校验方

法，将逻辑校验方法设计在输入程序中，由计算机自动校验。例如，输入日期时，计算机马上进行逻辑性检查，包括年月日是否大于0，月份是否在1～12之间等。

(4) 金额计算校验：指在凭证输入的过程中，由计算机程序自动根据有关数据进行一次金额计算，再与输入的金额进行核对的一种检验方法。例如，一张凭证中有数量、单价、金额等数据，当输入了数量、单价后，计算机自动计算出金额，如果与输入的金额不一致，则输入错误。

(5) 平衡校验：采用借贷记账法，记账规则是"有借必有贷，借贷必相等"。利用这种平衡关系，可在每张凭证数据输入时，由计算机程序自动进行借贷金额平衡校验。若借方金额等于贷方金额，方可进行下一步处理，否则数据不对，就输出错误信息。

(6) 校验位校验：根据已编好的数码，通过一定的数学模型，求得一位数字加在代码后面作为校验位，以验证输入代码的正确性。例如，第二代身份证号码的最后一位数字就是校验位。

(7) 控制总数校验：工作人员先用手工求出数据的总值，然后在数据输入过程中由计算机程序累计总值，将二者对比校验。

(8) 数据类型校验：校验是数字型还是字符型或其他符合要求的类型。

(9) 格式校验：校验数据记录中各数据项的位数和位置是否符合预先规定的格式。例如，姓名列规定为18位，而姓名的最大位数是17位，则该列的最后一位一定是空白。该位若不是空白，就认为该数据项错位。

(10) 顺序校验：检查记录的顺序，例如，要求输入数据没有缺号时，通过顺序校验可以发现被遗漏的记录。又如，要求记录的序号不得重复时，即可查出有无重复记录。

7.2　输出设计

在软件实现过程中，输出设计占据重要地位。因为计算机系统对输入数据进行加工处理的结果只有通过输出才能让用户使用，故输出的内容与格式是用户最关心的问题。另一方面，从系统开发的角度来看，输入信息只有根据输出要求才能确定，即输出决定输入。

一般对输出信息的基本要求是准确、及时、适用。输出设计的详细步骤包括确定输出类型与输出内容、确定输出方式(设备与介质)、表格设计等。输出信息直接服务于用户，在设计过程中，系统设计员应深入了解用户的信息要求，与用户充分协商。

7.2.1　设计内容

输出设计的内容包括以下几个方面：

(1) 有关输出信息使用方面的内容，包括信息的使用者、使用目的、报告量、使用周期、有效期、保管方法和输出份数等。

(2) 输出信息的内容，包括输出项目、位数、数据形式(文字、数字)。

(3) 输出方式，如表格、图形或文件。最常用的输出方式有两种：一种是报表；另一种是图形。究竟采用哪种输出方式为宜，应根据系统分析和管理业务的要求而定。一般来说，对于基层或具体事务的管理者，应用报表方式给出详细的记录数据；而对于高层领导或宏观、综合管理部门，则应该使用图形方式给出比例或综合发展趋势的信息。

(4) 输出设备,如打印机、显示器、卡片输出机等。

(5) 输出介质,如输出到磁盘还是磁带上,输出用纸是专用纸还是普通纸等。

输出设备和介质如表 7.1 所示。

表 7.1 输出设备和介质一览表

输出设备	打印机	卡片或纸带输出机	磁带机	磁盘机	终端	绘图仪	缩微胶卷输出机
介质	纸张	卡片或纸带	磁带	磁盘	屏幕	图纸	缩微胶卷
用途和特点	便于保存,费用低	可作其他系统输入之用	容量大,适于顺序存取	容量大,存取、更新方便	响应灵活的人机对话	精度高,功能全	体积小,易保存

7.2.2 报表方式输出

报表是一般系统中用得最多的信息输出工具。报表通常覆盖整个组织的软件系统,输出报表的种类都在百种以上,这样庞大的工作量对系统开发工作的压力是很大的。所以在实际工作中,经常是确定了报表的种类和格式之后,开发出一个报表模块,并由它来产生和打印所有报表。报表模块原理如图 7.1 所示。

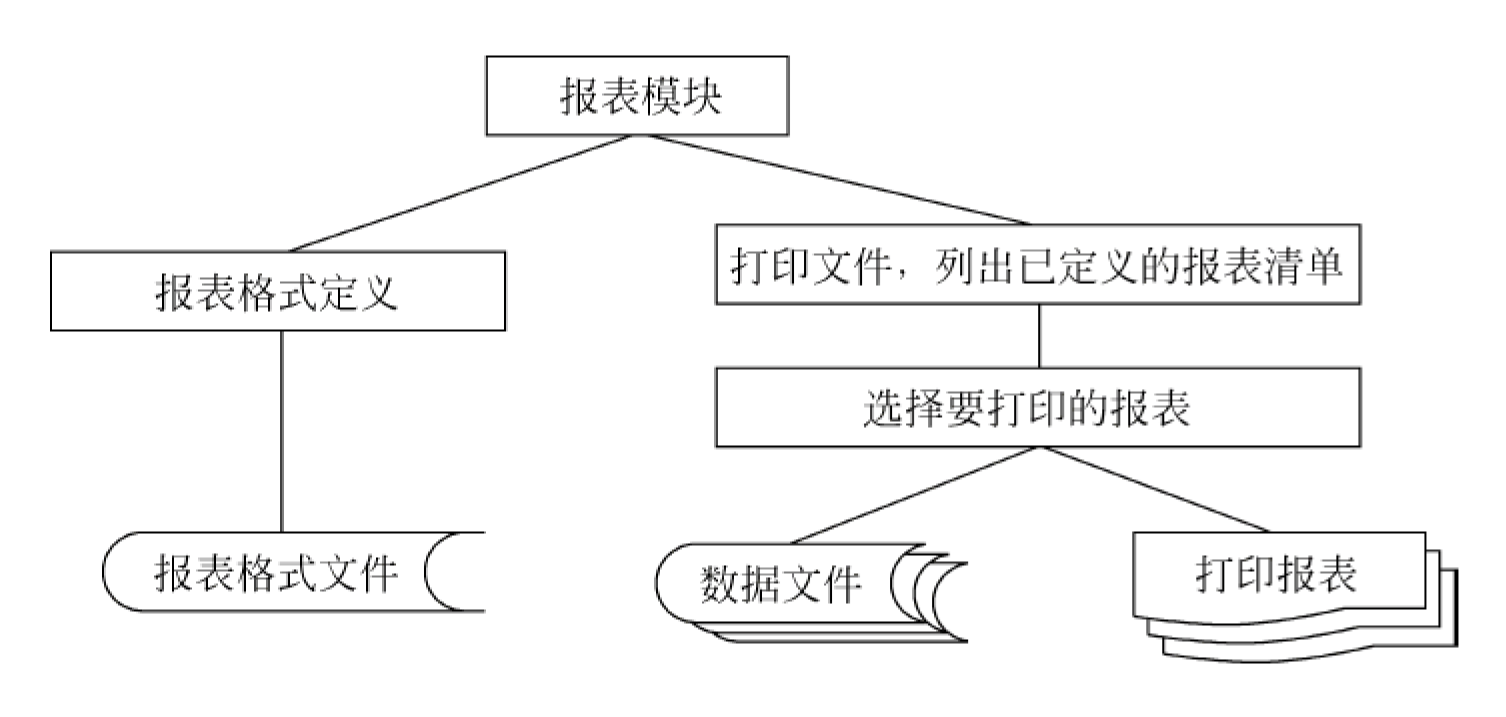


图 7.1 报表模块原理

图 7.1 分为两部分:左边是定义一个报表的格式部分,定义完后将其格式以一个记录的方式存于报表格式文件中;右边是打印报表部分,首先打开文件,读出已定义的报表各列显示在弹出式菜单中,待用户选择,当用户选中某个报表后,系统读出该报表的格式并对数据进行打印。

目前并没有统一的报表类型分类标准,从内容上把报表划分为如下四类。

(1) 明细表:用来反映在确定的时间范围内事务活动的详细情况,强调对信息反映的翔实性。在明细表中也会出现一些冗余信息和汇总信息。

(2) 汇总表:反映业务活动的综合信息。组织中的不同人员对汇总表信息有不同的要求,一般在软件开发阶段很难设计出用户需要的所有汇总表,实际上很多汇总表是在系统运行过程中,根据用户的需要临时生成的。因此,软件应提供由用户自己设计汇总表格式,并提取汇总数据的功能。

(3) 分析表:反映信息的对比和分析情况。根据报表中信息的详细程度,可分为明细

分析表和汇总分析表两种形式，但一般是汇总分析表较多。

(4) 历史表：用来反映业务活动的历史记录。历史表并不是一种单一的报表类型，可以采用明细表、汇总表、分析表的形式。在历史表中主要反映过去的信息。

7.2.3　图形方式输出

就目前的计算机技术来说，将系统的各类统计分析结果用图形方式输出已经是一件很容易的事了。大多数编程软件都提供了绘图工具或图形函数等，例如 C 语言、Lotus、FOXGRAPH 等，利用这些工具就可以产生系统需要的图形。但是使用这些工具绘图要求开发者具有一定的技术基础，并且开发工作量较大。比较简单的，可以借用 Excel 来产生各种分析图形，具体方法如图 7.2 所示。

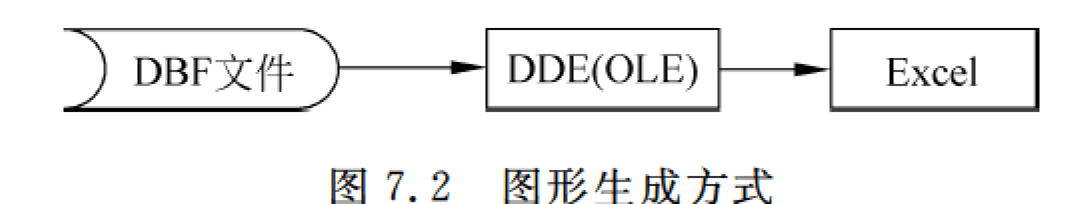


图 7.2　图形生成方式

如果系统是以 VFP 等为主要语言编写的，则可以利用 Excel 的动态数据交换(Dynamic Data Exchange，DDE)功能或者对象连接与嵌入(Object Link and Embed，OLE)，借用 Excel 来完成统计分析和图形输出功能。这样，熟练者很快就可完成很多种统计分析的图形。

一般来讲，图形方式比报表更直观。常用的图形类型有以下四类。

(1) 散点图：可以反映数据变化的规律和趋势，如图 7.3 所示。在企业管理过程中，通常用散点图反映业务过程的历史数据，然后通过散点图来预测业务未来的变化趋势。

(2) 折线图：用来反映一定时间区间内数据变化的波动情况，如图 7.4 所示。折线图也可以表现数据的变化趋势，但与散点图的区别是折线图增加了时间维数，因此能够表现出数据随时间变化的趋势。折线图可以用来比较在相同时间范围内两个或多个业务的变化情况。它可以表现产品销售业务、销售人员的销售活动、学生对课程的登记情况等业务活动。需要注意的是，折线图中 X 轴通常用来表示时间，而 Y 轴表示业务值。

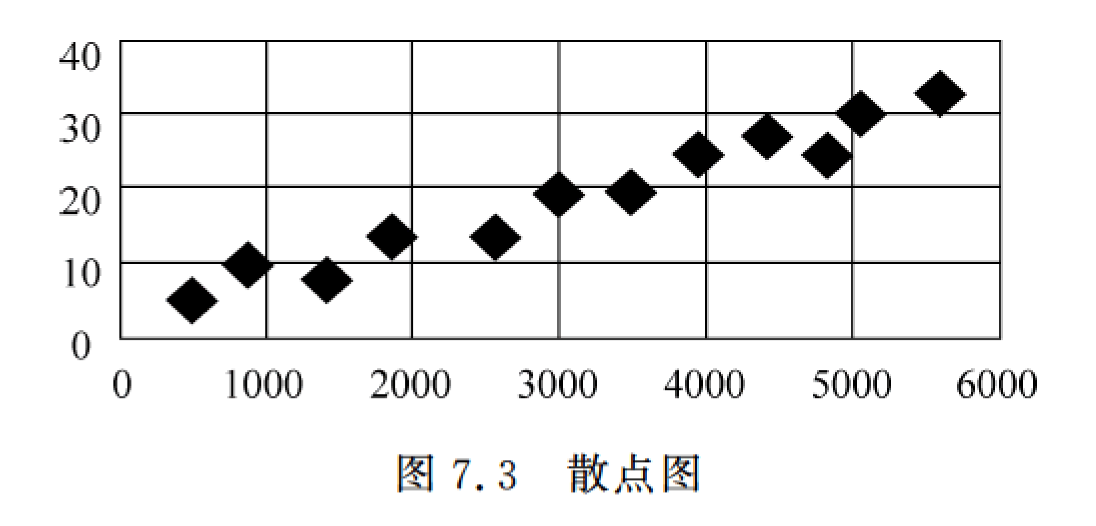


图 7.3　散点图

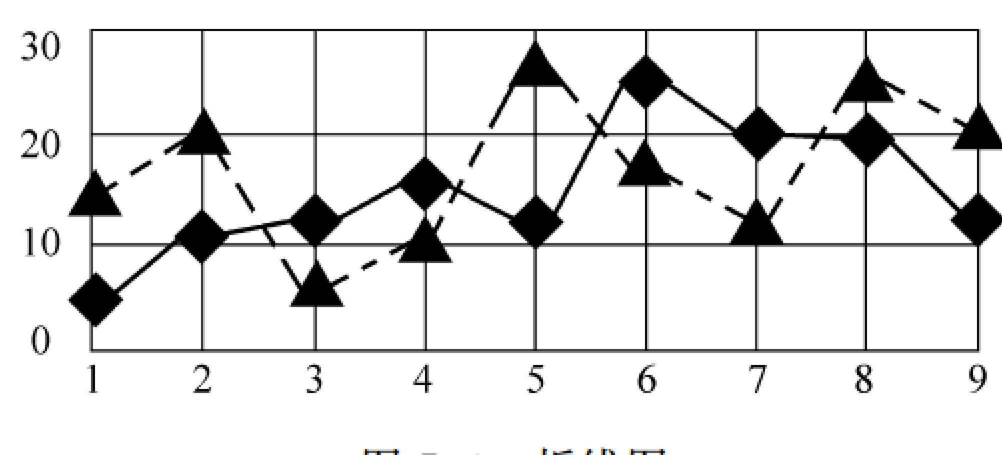


图 7.4　折线图

(3) 条形图：用来表示各分量之间的关联关系和比例关系。按照图中条形棒的方向，可以把条形图分为水平条形图和垂直条形图。水平条形图用来对相同时间区间内的不同项目进行比较，而垂直条形图用来对不同时间区间内的同一项目进行比较。图 7.5 就是垂直条形图的例子。条形图的缺点是不能反映相同时间区间内所有项目的合计和不同时间区间内同一项目的合计。

(4) 圆饼图：通过圆和多个扇面来表示整体和部分以及各部分在整体中所占的比例，

如图 7.6 所示。圆饼图可以设计成多种不同的形式,如可以设计成如图 7.6 所示的二维结构,也可以设计成三维结构。为了突出其中一个部分或多个部分,可以在图中突出某一个扇形或多个扇形。另外,在每一个扇形中也可以标出所表示的比例数字。

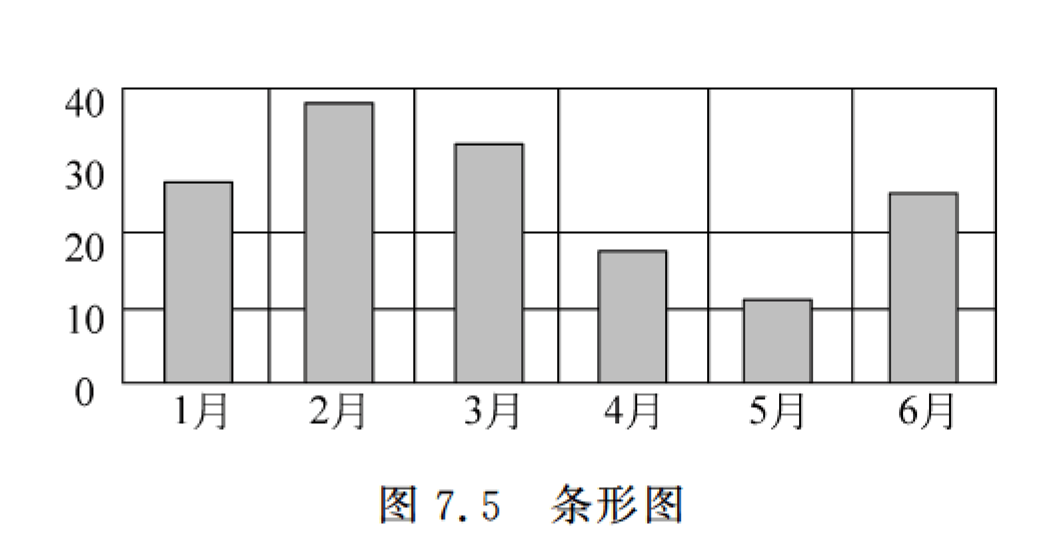


图 7.5 条形图

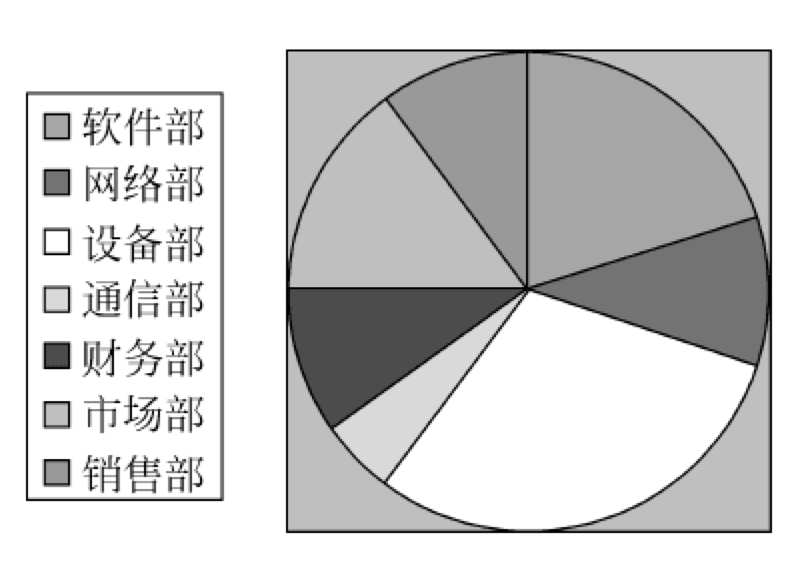


图 7.6 圆饼图

7.3 屏幕界面设计

屏幕界面是系统与用户之间的接口,也是控制和选择信息输入输出的主要途径。屏幕界面设计应坚持友好、简洁、实用、易于操作等原则,尽量避免过于烦琐和花哨。

7.3.1 设计规则

Ben Shneiderman 经过大量实践,总结出屏幕界面设计的八条规则,在《*Designing the User Interface*》一书中进行了阐述。Shneiderman 的八条经典规则被称为"黄金规则",是屏幕界面设计的最佳指南。这八条规则来源于 Shneiderman 二十多年的经验,并经过精炼、改进和延伸,适用于大多数的交互式系统,但对于特定的设计领域需要验证和调整。

(1) 尽可能保持一致性。一致性包括类似的操作环境应提供一致的操作序列;相同的术语应该用在提示、菜单和帮助里;颜色、布局、大小写、字体等应当自始至终保持一致。一致性是人们习惯的需要,既可以保持界面的规整、简洁,又可以减轻人们学习和使用软件系统的负担。

(2) 为熟练用户提供快捷键。快捷键能够提高系统的操作速度。对于初次使用软件系统的人员,提供丰富的界面说明和联机帮助,并更多地使用菜单和选项。但对于经常使用系统的熟练用户,过多的界面操作反而增加操作量,降低使用效率。

(3) 提供丰富的反馈信息。对用户的每一个操作都有对应的系统反馈信息,以便使用户了解对操作的确认。如果长时间得不到系统反馈,用户将无从知道操作的正确性。对于常用的或较次要的操作,反馈信息可以很简短;而对于不常用但很重要的操作,反馈信息就应丰富一些。对象的可视化实现可以方便清晰地显示出这种变化。

(4) 设计完整的对话过程。用户处理每一个业务都是一个完整的对话过程,因此,系统设计的每一个对话过程应该是完整的,有开始部分、中间处理部分和结束部分。对话过程可以使用户明确当前进行的操作,以及系统准备接受或进行的下一步操作。

(5) 提供错误预防机制。应当尽可能地设计不让用户犯严重错误的系统,但用户在输入数据、按键、操作顺序等方面的错误又是不可避免的。如果用户犯了错误,界面应当检测

到错误，并提供简单的、有建设性的、具体的指导来帮助恢复错误。错误的操作应该让系统状态保持不变，或者界面应当提供关于恢复状态的说明。

(6) 允许轻松的反向操作。操作应尽可能地允许反向。这个特点可以减轻用户的焦虑，由于用户知道错误可以被撤销，这就会鼓励用户尝试不熟悉的选项。反向操作单元可以是单独操作、单个数据输入任务或一组完整操作。

(7) 支持内部控制轨迹。用户在操作过程中，如果系统能够随时把控制的内部轨迹提示给用户，用户会感到自己一直在控制着系统，能够了解系统的工作过程。这样还能鼓励用户成为行为的主动者，而不是行为的响应者。

(8) 减少短时记忆负担。由于人凭借短时记忆进行信息处理存在局限性(由经验法则可知，人可以记忆 5～9 个信息块)，所以要求显示简单、多页显示统一以及窗口移动频率低，并且要保证分配足够的时间，用于学习代码、记忆操作方法和操作序列。另外，还应该提供对命令语法、缩略语、代码以及其他信息进行适当的在线访问。

7.3.2　设计要素

界面设计是为了满足软件专业化、标准化的需要而产生的对软件使用界面进行美化、优化、规范化的设计分支。具体包括软件启动封面设计、软件框架设计、按钮设计、面板设计、菜单设计、标签设计、图标设计、滚动条及状态栏设计、安装过程设计、包装及商品化等。在设计过程中应注意的关键问题如下：

(1) 启动封面设计。启动封面最终要设计成为高清晰度的图像，如果软件启动封面需要在不同的平台、操作系统上使用，要考虑转换不同的格式，并且选用的色彩不宜超过 256 色，最好为 216 安全色。软件启动封面大小多为主流显示器分辨率的 1/6。如果是系列软件，将考虑整体设计的统一性和延续性。在上面应该醒目地标注制作或支持的公司标志、产品商标、软件名称、版本号、网址、版权声明、序列号等信息，以树立软件形象，方便使用者或购买者在软件启动时得到提示。插图宜使用具有独立版权的、象征性强的、识别性高的、视觉传达效果好的图形，若使用摄影也应该进行数字处理，以形成软件的个性化特征。

(2) 框架设计。软件框架设计非常复杂，因为涉及软件的使用功能，设计师应该对软件产品的程序和使用比较了解，这就需要设计师有一定的软件跟进经验，能够快速地学习软件产品，并且和程序开发人员及程序使用人员进行沟通交流，以设计出友好的、独特的、符合程序开发原则的软件框架。软件框架设计应该简洁明快，尽量少用无谓的装饰，应该考虑节省屏幕空间、各种分辨率的大小、缩放时的状态和原则，并且为将来设计的按钮、菜单、标签、滚动条及状态栏等预留位置。设计时对整体色彩组合进行合理搭配，将软件商标放在显著位置，主菜单应放在左边或上边，滚动条放在右边，状态栏放在下边，以符合视觉流程和用户使用心理。

(3) 按钮设计。软件按钮设计应该具有交互性，即应该有 3～6 种状态效果：点击时的状态、鼠标放在上面未点击时的状态、点击前鼠标未放在上面时的状态、点击后鼠标未放在上面时的状态、不能点击时的状态、独立自动变化的状态。按钮应具备简洁的图示效果，应能够让使用者产生功能关联反应，群组内按钮应该风格统一，功能差异大的按钮应该有所区别。

(4) 面板设计。软件面板设计应该具有缩放功能,面板应该对功能区间划分清晰,应该和对话框、弹出框等风格匹配,尽量节省空间,切换方便。

(5) 菜单设计。菜单设计一般有选中状态和未选中状态,左边应为名称,右边应为快捷键,如果有下级菜单应该有下级箭头符号,不同功能区间应该用线条分割。菜单可分为下拉式菜单和弹出式菜单两种类型。下拉式菜单是一种应用于主控界面的菜单类型,一般分为两层结构:第一层为主菜单,各个选项的名称按水平方向排成一行,被固定放在窗口最上方的一个带形区域中;第二层为主菜单的各个选项的子菜单,子菜单按垂直方向排列,每个子菜单放置在其对应的主菜单项的下方,平时各个子菜单被隐藏起来,只有当单击主菜单项时,对应的子菜单才被弹出。弹出式菜单是垂直排列功能选项的矩形框,可被下拉式菜单或其他窗口功能选项驱动弹出,可以是单层结构或多层结构,位置可以根据用户操作或当时的操作环境确定。

(6) 标签、文本框、列表框、复选框设计。标签用来在窗口中显示一段不能编辑的文本,使用标签可以对文本框、列表框等控件进行解释或描述,也可以在窗口中输出一段说明性文字信息,还可向用户输出提示、出错等信息;文本框是用来接收用户输入信息的正文编辑区域,用户可以在文本框中的光标位置输入信息,文本框可以分为单行和多行,输入内容超出编辑框宽度时,可自动滚动;列表框是向用户提供功能、信息或参数的选项列表,进入列表框后,光条显示在列表框的第一个选项上面,用户可以把光条移动到所要选择的项上;复选框表示对某个选项是否选择,用一个小方框表示,如果选中所表示的选项,则复选框显示一个小对号,没有选中则不显示小对号。

(7) 图标设计。图标设计色彩不宜超过 64 色,大小为 16×16、32×32 两种。图标设计是方寸艺术,应该加以着重考虑视觉冲击力,需要在很小的范围表现出软件的内涵,所以很多图标设计师在设计图标时使用简单的颜色,利用对色彩和网点的空间混合效果,做出精彩图标。

(8) 滚动条及状态栏设计。滚动条主要是对固定大小区域性空间中内容的变换进行设计,应该有上下箭头、滚动标等,有些还有翻页标。状态栏的功能是对软件当前状态的显示和提示。

(9) 安装过程设计。安装过程设计主要是将软件安装的过程进行美化,包括对软件功能进行图示化。

(10) 包装及商品化。软件产品的包装应该考虑保护好软件产品,功能的宣传融合于美观中,可以印刷部分产品介绍、产品界面设计等。

7.3.3 设计内容

界面设计的第一步是界面结构设计,将任务设计的结果作为输入,设计成一组逻辑模块,然后加上存取机制,把这些模块组织成界面结构。存取机制可以是分层的、网络的或直接的,机制的类型主要由任务结构决定,也取决于设计风格。例如,菜单提供了层次结构;图标则是直接存取,也可以是层次的;而命令语言可提供网络也可提供直接存取机制。第二步是界面细化设计,将每一模块分成若干步,每一步又被组装成细化的对话设计。界面细化设计流程如图 7.7 所示。

界面设计内容主要包括以下三个方面。

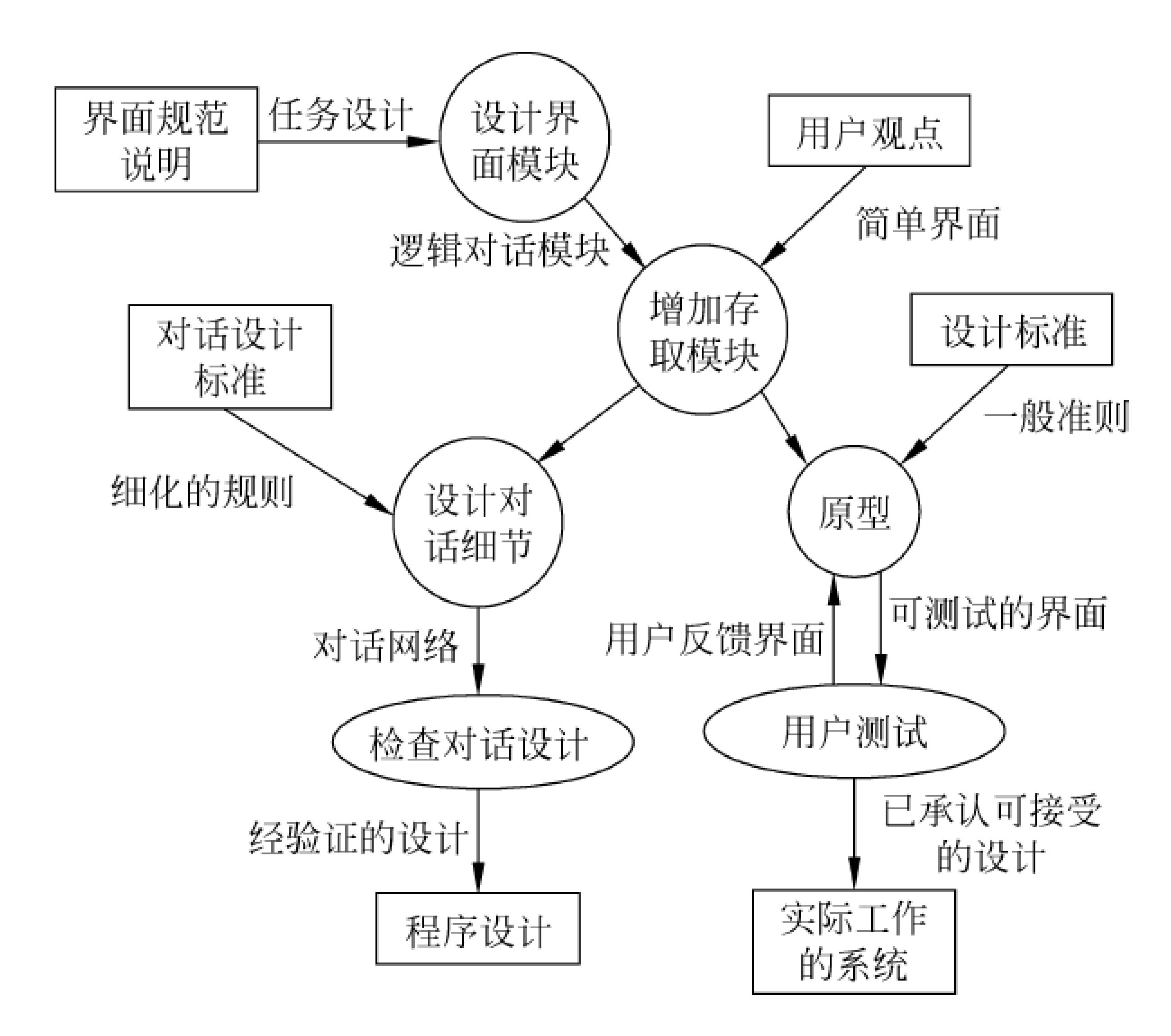


图 7.7　界面细化设计流程图

1. 界面对话设计

用户与软件系统之间的交互过程实际是一个对话过程，用户通过软件完成一个完整功能，需要与系统发生一次对话过程。在界面设计中要使用对话风格的选择，并加上用户存取和控制机制。对话是以任务顺序为基础，但要遵循如下原则：

(1) 反馈(feed back)。随时将正在做什么的信息告知用户，尤其是在响应时间较长的情况下。

(2) 状态(status)。告诉用户正处于系统的什么位置，避免用户在错误环境下发出了语法正确的命令。

(3) 脱离(escape)。允许用户中止一种操作，且能脱离该选择，避免用户死锁发生。

(4) 默认值(default)。只要能预知答案，尽可能设置默认值，节省用户工作。

(5) 简化(predigst)。尽可能简化对话步骤，使用省略语或代码来减少用户击键数。

(6) 求助(help)。尽可能提供联机在线帮助。

(7) 复原(undo)。在用户操作出错时，可返回并重新开始。

在对话设计中应尽可能考虑上述准则，媒体设计对话框有许多标准格式供选用。另外，对界面设计中的冲突因素应进行折中处理。

2. 数据输入界面设计

数据输入界面往往占终端用户的大部分使用时间，也是计算机系统最易出错的部分之一。数据输入界面设计的总体目标是简化用户工作，并尽可能降低输入出错率，也要允许用户输入错误。

数据输入界面设计可采用的方法和策略如下：

(1) 尽可能减轻用户记忆，采用列表选择。对共同输入内容设置默认值，使用代码和缩写等，系统自动填入用户已输入过的内容。

(2) 使界面具有预见性和一致性。用户应能控制数据输入顺序并使操作明确,采用与系统环境(如 Windows 操作系统)一致风格的数据输入界面。

(3) 防止用户出错。在设计中可采取确认输入(只有用户按下键,才确认),明确地移动(使用 Tab 键或鼠标在窗口中移动),明确地取消,已输入的数据并不删除。对删除必须再一次确认,对致命错误,要警告并退出。对不太可信的数据输入,要给出建议信息,处理不必停止。

(4) 提供反馈。使用户能查看已输入的内容,并提示有效的输入回答或数值范围。

(5) 按用户速度输入和自动格式化。用户应能控制数据输入速度,系统能进行自动格式化。

(6) 允许编辑。理想的情况是在输入后能允许编辑,且采用风格一致的编辑格式。

数据输入界面可通过对话设计方式实现,若条件具备尽可能采用自动输入。特别是图像、声音输入,在远程输入及多媒体应用中会迅速发展。

3. 屏幕显示设计

屏幕显示设计主要包括布局(layout)、文字与用语(message)及颜色(color)等。

1) 布局

屏幕布局因功能不同考虑的侧重点不同。各功能区要重点突出,功能明显。无论哪一种功能设计,屏幕布局都应遵循如下五项原则:

(1) 平衡原则。注意屏幕上下左右平衡。不要堆挤数据,过分拥挤的显示会产生视觉疲劳和接收错误。

(2) 预期原则。屏幕上所有对象,如窗口、按钮、菜单等处理应一致化,使对象的动作可预期。

(3) 经济原则。在提供足够信息量的同时还要注意简明、清晰。特别是媒体,要运用好媒体选择原则。

(4) 顺序原则。对象显示的顺序应依需要排列。通常最先出现对话,然后通过对话将系统分段实现。

(5) 规则化原则。画面应对称,显示命令、对话及提示行等,在一个应用系统的设计中尽量统一规范。

2) 文字与用语

文字与用语除作为正文显示媒体时出现外,还在设计题头、标题、提示信息、控制命令、会话等功能时展现。对文字与用语设计的格式和内容注意如下:

(1) 用语简洁性。避免使用计算机专业术语;尽量用肯定句而不用否定句;用主动语态而不用被动语态;用礼貌而不过分的强调语句进行文字会话;对不同的用户,实施心理学原则使用用语;英文词语尽量避免缩写;在按钮、功能键标示中应尽量使用描述操作的动词;在有关键字的数据输入对话和命令语言对话中,采用缩略码形式;文字较长时,可用压缩法减少字符数或采用一些编码方法。

(2) 格式。在屏幕显示设计中,一幅画面不要文字太多,若必须有较多文字时,尽量分组分页,在关键词处进行加粗、变字体等处理,但同行文字尽量字形统一。英文单词除了标

语以外，尽量采用小写和易认的字体。

(3) 信息内容。信息内容显示不仅采用简洁、清楚的表达，还应采用用户熟悉的简单句子，尽量不用左右滚屏。内容较多时，应以空白分段或以小窗口分块，以便记忆和理解。重要字段可用粗体或闪烁等，吸引注意力和强化效果，强化效果有很多种，针对实际进行选择。

3) 颜色的使用

颜色的调配对屏幕显示也是重要的一项设计，颜色除是一种有效的强化技术外，还具有美学价值。使用颜色时应注意以下几点：

(1) 限制同时显示的颜色数。一般同一画面中不宜超过四种或五种颜色，可用不同层次及形状来配合颜色，增加变化。

(2) 画面中活动对象颜色应鲜明，而非活动对象颜色应暗淡。对象颜色应尽量不同，前景颜色宜鲜艳一些，背景颜色则应暗淡。

(3) 尽量避免不兼容的颜色放在一起，如黄与蓝、红与绿等，除非作为对比使用。

(4) 若用颜色表示某种信息或对象属性，要使用户懂得这种表示，且尽量用常规准则。

总之，屏幕显示设计最终应达到令人愉悦的显示效果，要指导用户注意到最重要的信息，但又不包含过多的相互矛盾和刺激的内容。

7.4 程序设计语言

7.4.1 语言分类

随着计算机技术的发展，目前已经出现了数百种程序设计语言，但被广泛应用的只有几十种。由于不同种类的语言适用于不同的问题域和系统环境，因此了解程序设计语言的分类可以帮助软件工程师选择合适的语言。通常可将程序设计语言分为面向机器语言和高级语言两大类。

面向机器语言包括机器语言(machine language)和汇编语言(assemble language)两种。

高级语言中的语句标识符与人类的自然语言(英文)较为接近，并且采用了人们十分熟悉的十进制数据表示形式，利于学习和掌握。高级语言的抽象级别较高，不依赖于实现它的计算机硬件，且编码效率较高，往往一条高级语言语句对应着若干条机器语言或汇编语言指令。高级语言程序需要经过编译或解释之后，才能生成可在计算机上执行的机器语言程序。高级语言按应用特点的不同，可分为通用语言和专用语言两大类。

从软件工程的角度，根据程序设计语言发展的历程，可以大致分为四类。

1. 从属于机器的语言(第一代语言)

从属于机器的语言是由机器指令代码组成的语言，对于不同的机器就有相应的一套机器语言。用这种语言编写的程序都是二进制代码，且所有的地址分配都是以绝对地址的形式处理。存储空间的安排，寄存器、变址的使用，都由程序员编程来确定。因此使用机器语言编写的程序很不直观，在计算机内的运行效率很高，但编写出的机器语言程序出错率也较高。

2. 汇编语言(第二代语言)

汇编语言比机器语言直观,每一条符号指令与相应的机器指令有对应关系,同时又增加了一些诸如宏、符号地址等功能。存储空间的安排可由机器解决。不同指令集的处理器系统有相应的汇编语言。从软件工程的角度来看,汇编语言只是在高级语言无法满足设计要求时,或者不具备支持某种特定功能(例如特殊的输入输出)的技术性能时,才被使用。

3. 高级程序设计语言(第三代语言)

传统的高级程序设计语言如 FORTRAN、COBOL、ALGOL、BASIC 等曾得到广泛应用,目前都已有多种版本。有的语言得到较大改进,甚至形成了可视化开发环境,具有图形设计工具、结构化的事件驱动编程模式、开放的环境,用户可以既迅速又简便地编制出 Windows 下的各种应用程序。

通用的结构化程序设计语言具有很强的过程功能和数据结构功能,并提供结构化的逻辑构造。这一类语言的代表是 PL/1、PASCAL、C 和 Ada 等。此外,COBOL 78、Turbo BASIC 等也应归入到第三代程序设计语言的范围。

专用语言是为特殊应用而设计的语言。通常具有特殊的语法形式,面对特定问题,输入结构及词汇表与该问题的相应范围密切相关。有代表性的专用语言有 APL、Lisp、Prolog、Smalltalk、C++、FORTH 等。从软件工程的角度来看,专用语言支持了特殊应用,将特定的设计要求翻译成可执行代码。但是可移植性和可维护性比较差。

4. 第四代语言(4GL)

4GL(Fourth-Generation Language)用不同的文法表示程序结构和数据结构,是在更高一级抽象的层次上表示这些结构,不再需要规定算法的细节。4GL 兼有过程性和非过程性两重特性。程序员规定条件和相应的动作是过程性部分;指出想要的结果是非过程性部分。然后由 4GL 语言系统运用专门领域的知识来填充过程细节。

Martin 把第四代语言分为以下几种类型:

(1) 查询语言。用户可利用查询语言对预先定义在数据库中的信息进行较复杂的操作。

(2) 程序生成器。只需很少的语句就能生成完整的第三代语言程序,不必依赖预先定义的数据库。

(3) 其他 4GL。如判定支持语言、原型语言、形式化规格说明语言等。

7.4.2 语言特性

程序设计语言具有心理、工程和技术三大特性。

1. 心理特性

程序体现编程者解决问题的思路,不同的人有不同的解决问题的思路,同一个人在不同心理状态下的解决问题的思路往往也会有所不同。所谓程序设计语言的心理特性,就是指能够影响编程者心理的语言性能。这种影响主要表现在以下方面:

(1) 歧义性。歧义性是指程序设计语言中的某些语法形式使不同的人产生不同的理解。如C语言中的表达式 a/c * b 有人理解为(a/c) * b,有人却理解为 a/(c * b)。当然,这只是由于某些人对语言中某些语法规则不了解导致的,对于语言编译系统来说只有确定的一种解释。

(2) 简洁性。简洁性是指编程者要使用该语言必须记住的各种语法规则(包括语句格式、数据类型、运算符、函数定义形式等)的信息量。需记忆的信息量越大,简洁性越差,人们掌握起来也就越难。但如果程序设计语言的语法成分太少,过于简洁,又会给阅读程序带来麻烦,不利于人的理解。因此,对于一个好的程序设计语言来说,既要具有一定的简洁性,又要具有较高的可理解性。

(3) 局部性和顺序性。局部性是指语言的联想性,也就是相关内容的相对集中性。在编程过程中,将实现某一功能的语句集中书写在一个模块中,由模块组装成完整的程序,并要求模块具有高内聚、低耦合的特点,目的就是希望加强程序的局部性。顺序性是指语言的线性特征。例如,对于顺序结构的程序人们很容易理解,如果程序中存在大量的分支结构和循环结构,理解起来就比较困难了。语言的局部性和顺序性是由人类习惯于用联想的方式,即按逻辑上的线性序列记忆事物的特性所决定的,局部性和顺序性的加强可提高程序的可理解性。

2. 工程特性

程序设计语言所应具备的工程特性主要体现在以下几个方面:

(1) 可移植性。可移植性反映了程序在不同机器环境下的通用性和适应性。不同机器环境包括不同的机型、不同的操作系统版本及不同的应用软件包。若一个程序不加修改或稍加修改就可以应用于不同的机型、运行应用于高版本的操作系统或集成到不同的应用软件包中,则称这个程序具有较高的可移植性。

(2) 语言编译器的实现效率。不同语言的编译器在将源程序代码翻译成目标代码的过程中,由于编译程序设计质量的不同,导致生成的目标代码大小和执行效率不尽相同。为了获得高效率的目标代码,选择语言时应充分考虑语言编译器的实现效率。

(3) 开发工具的支持。为了缩短编码阶段花费的时间以及提高编码质量,应选择具有良好开发工具支持的程序设计语言。这些开发工具主要包括编译程序、连接程序、交互式调试器、交叉编译器、图形界面及菜单系统生成程序、宏处理程序等。

(4) 可维护性。程序维护是软件工程活动的一项重要内容。为了提高程序的可维护性,即方便对源程序的修改,程序中采用的语言必须具有良好的可读性和易于使用等特点。

3. 技术特性

在确定了软件开发项目需求后,根据项目选择具有相应技术特性的程序设计语言,对保证软件质量具有非常重要的作用。不同的语言具有不同的技术特性。

例如,有的语言提供了丰富的数据类型或复杂的数据结构;有的语言具有很强的实时处理能力;有的语言可方便地实现大量数据的查询及增加、删除、修改等功能。

根据语言的技术特性为项目选择合适的程序设计语言,不但可以使编写的程序很好地

满足项目要求，而且对后期测试和维护工作也非常有益。

7.4.3 语言选择

选择语言时不能只考虑理论标准，同时要兼顾实用标准。下面分别简要地对选择语言的主要理论标准和实用标准进行介绍。

1. 理论标准

理论标准包括以下两个方面：

(1) 理想的模块化机制、易于阅读和使用的控制结构及数据结构。

(2) 完善、独立的编译机制。完善的编译系统应尽可能多地发现程序中的错误，便于程序调试和提高软件可靠性，并且可以使生成的目标代码紧凑、高效。独立的编译机制便于程序开发、调试和维护，可以降低软件开发和维护成本。

2. 实用标准

实用标准包括以下几个方面：

(1) 软件用户的要求。由于用户是软件的使用者，因此软件开发者应充分考虑用户对开发工具的要求。特别是当用户负责软件维护工作时，通常要求采用用户熟悉的语言进行编程。

(2) 软件规模。语言系统的选择与软件规模有直接关系。开发小的计算程序可选择 C 语言、BASIC 语言等；开发较大的应用程序可考虑.NET 或 Java 语言。如果软件规模非常庞大，为了提高开发的效率和质量，可能考虑选择几种语言分别开发不同的模块。

(3) 软件运行环境。软件提交给用户后，将在用户的机器上运行，在选择语言时应充分考虑用户运行软件环境对语言的约束。此外，运行目标系统的环境中可以提供的编译程序往往也限制了可以选用语言的范围。

(4) 软件开发方法。有时，编程语言的选择依赖于开发方法。如果用快速原型模型开发，要求能快速实现原型，因此宜采用 4GL；如果是面向对象开发，宜采用面向对象的语言编程。

(5) 可以得到的软件开发工具。由于受开发经费制约，往往使开发人员无法任意选择、购买合适的正版软件开发工具。此外，若能选用具有支持该语言程序开发的软件工具，将有利于目标系统的实现和验证。

(6) 软件开发人员的知识。软件开发人员采用自己熟悉的语言进行开发，可以充分运用积累的经验，使开发的目标程序具有更高的质量和运行效率，并可以大大缩短编码阶段的时间。为了能够根据具体问题选择更合适的语言，软件开发人员应拓宽知识面，多掌握几种程序设计语言。

(7) 软件的可移植性。要使开发出的软件能适应于不同的软、硬件运行环境，应选择具有较好通用性的、标准化程度高的语言。

(8) 软件的性能要求。不同语言开发的程序具有不同的性能。如果对执行速度要求比较高，可考虑 C 语言或 Visual C++语言；如果对执行速度要求特别高，就要考虑汇编语言；如果对执行速度要求不高，可采用开发速度较快的 Visual Basic 语言。

(9) 软件的应用领域。任何语言编译系统设计的出发点都有所不同,对某一领域问题的处理能力也就存在较大差异,因此不存在真正适用于任何应用领域的语言,通用语言也不例外。例如,FORTRAN 语言最适用于工程科学计算,Java 语言最适用于处理网络编程中的问题。所以,选择语言时一定要充分考虑软件的应用领域。

7.5 编程风格

在过去很长一段时间,人们始终认为,程序是给计算机执行的,只要程序逻辑正确,计算机能执行,完成指定的功能就行了,将程序风格看得无关紧要。

随着软件规模和复杂性的增加,人们才认识到程序的易读性和程序设计风格的重要性,人们在维护、调试、测试程序时经常需要反复地阅读程序,甚至阅读程序的时间比编写程序的时间还要多。不仅程序设计者本人要多次阅读,开发小组的其他设计人员和维护人员也要反复阅读。阅读程序是软件开发和维护过程中的一个重要组成部分。程序员在编写程序时,应当意识到今后会有人反复阅读这个程序,并沿着各自的思路去理解程序的功能。

在编写程序时多花些工夫,讲究程序设计风格,将大大减少人们阅读程序的时间,从整体上看是提高了效率。因此,良好而规范地表示程序的逻辑结构,是软件设计人员应具有的程序设计风格。

从阅读的角度来看,程序实际上也是一种供人阅读的文章。有些文章优雅、整洁,引人入胜,有些文章晦涩难懂,无法过目。这就是文章风格所起的作用。在编写程序中也存在程序设计风格的问题,应编写具有良好风格的程序。

影响程序设计风格的因素主要有三个方面:源程序文档化、标识符命名、语句构造与程序书写。

7.5.1 源程序文档化

源程序文档化是指程序中的说明性注释信息,使程序容易阅读。在程序中加入注释信息的目的是为了提高程序的可读性,为程序的测试和维护带来方便。几乎所有的程序设计语言都提供了专门用于书写注释信息的注释语句。为了使程序易于阅读和修改,应在必要的位置加上相应的注释。在修改程序时,不要忘记对相应的注释也要进行修改。

程序中的注释按用途可分为两类:序言性注释和功能性注释。

序言性注释位于模块的首部,用于说明模块的相关信息。序言性注释主要包括以下内容:对模块的功能、用途的简要说明;对模块的界面的描述,如调用语句的格式、各个参数的作用及需调用的下级模块清单等;对模块的开发历史的介绍,如模块编写者的资料、模块审核者的资料,以及建立、修改时间等;对模块的输入数据或输出数据的说明,如数据格式、类型及含义等。

功能性注释位于源程序模块内部,用于对某些难以理解语句段的功能,或某些重要标识符的用途等进行说明。通过在程序中加入恰当的功能性注释,可以提高程序的可读性和可理解性,对语句的注释应紧跟在被说明语句之后书写。

书写注释过程中应注意以下问题:

(1) 一般情况下,源程序有效注释量必须在20%以上。注释的原则是有助于对程序的阅读理解,在该加注释的地方一定要加注释,注释不宜太多也不能太少,注释语言必须准确、易懂、简洁。

(2) 边写代码边写注释,修改代码的同时修改相应的注释,以保证注释与代码的一致性。不再有用的注释要及时删除。

(3) 注释的内容要清楚、明了,含义准确,防止注释二义性。错误的注释不但无益反而有害。

(4) 避免在注释中使用缩写,特别是非常用缩写。在使用缩写时或使用之前,应对缩写进行必要的说明。

(5) 注释应与其描述的代码相近,对代码的注释应放在其上方或右方(对单条语句的注释)相邻位置,不可放在下面,如放于上方则需与其上面的代码用空行隔开。

(6) 对于所有的变量、常量,如果命名不能充分表达含义,在声明时都必须加以注释。变量、常量、宏的注释应放在其上方相邻位置或右方。

(7) 数据结构声明(包括数组、结构、类、枚举等)中,如果命名不是充分自注释的,必须加以注释。对数据结构的注释应放在其上方相邻位置,不可放在下面;对结构中每个域的注释放在此域的右方。

(8) 全局变量要有较详细的注释,包括对功能、取值范围、哪些函数或过程存取、存取时注意事项等的说明。

(9) 注释与所描述内容进行同样的缩排。可使程序排版整齐,并方便注释的阅读与理解。

(10) 对变量的定义和分支语句(条件分支、循环语句等)必须编写注释。这些语句往往是程序实现某一特定功能的关键,对于维护人员来说,良好的注释能帮助更好地理解程序,有时甚至优于设计文档。

(11) 对于Switch语句下的Case语句,如果因为特殊情况需要处理完一个Case后进入下一个Case处理,必须在该Case语句处理完后、下一个Case语句前加上明确的注释。

(12) 避免在一行代码或表达式的中间插入注释。除非必要,否则容易使代码的可理解性变差。

(13) 通过对函数或过程、变量、结构等正确的命名以及合理地组织代码结构,使代码成为自注释的。清晰准确的函数、变量等的命名可以增加代码可读性,并减少不必要的注释。

(14) 在代码的功能、意图层次上进行注释,提供有用、额外的信息。注释的目的是解释代码意图、功能和采用的方法,提供代码以外的信息,帮助读者理解代码,防止不必要的重复注释信息。

(15) 在程序块的结束行右方加注释标记,以表明某程序块的结束。当代码段较长,特别是多重嵌套时,这样做可以使代码更清晰,更便于阅读。

(16) 注释应考虑程序易读及外观排版的因素,使用的语言若是中、英文兼有的,建议多使用中文,除非能用非常流利准确的英文表达。注释语言不统一会影响程序易读性和外观排版,出于对维护人员的考虑,建议使用中文。

7.5.2 标识符命名

程序中的标识符包括模块名、变量名、常量名、标号名、子程序名、数据区名、缓冲区名等。标识符应能反映所代表的实际事务，应有一定的实际意义，能够见名知义，有助于理解程序的功能，增强程序的可读性。标识符命名的主要规则如下：

(1) 标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。标识符由多个单词构成时，每个单词的第一个字母最好采用大写或单词间用下划线分隔；标识符由较短的单词构成时，可通过去掉“元音”形成缩写；标识符由较长的单词构成时，可取单词的头几个字母形成缩写；一些单词用大家公认的缩写。

(2) 命名中若使用特殊约定或缩写，要有注释说明。应该在源文件的开始处，对文件中使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

(3) 为了便于程序的输入，标识符的名字不宜过长，通常不要超过八个字符。特别是对于那些对标识符长度有限制的语言编译系统，过长的标识符名没有任何意义。例如，在FORTRAN 77中，通常编译系统可以区分的标识符长度不超过六个字符。

(4) 为了便于区分，不同的标识符不要取过于相似的名字。例如student和students，很容易在使用或阅读时产生混淆。

(5) 自己特有的命名风格要自始至终保持一致，不可来回变化。个人的命名风格在符合项目组或产品组命名规则的前提下才可使用(即命名规则中没有规定到的地方才可有个人命名风格)。

(6) 对于变量命名，禁止取单个字符(如i、j、k、…)，建议除了要有具体含义外，还能表明变量类型、数据类型等，但i、j、k作局部循环变量是允许的。变量，尤其是局部变量，如果用单个字符表示，很容易写错(如将i写成j)，而编译时又检查不出来，有可能因为这个小小的错误而花费大量的时间查错。

(7) 命名规范必须与使用系统的风格保持一致，并在同一项目中使用统一风格。例如，采用UNIX的全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排方式。用做特殊标识，如标识成员变量或全局变量的m_和g_，其后加上大小写混排的方式是允许的。

(8) 尽量不要用数字或较奇怪的字符来定义标识符。

(9) 在同一软件产品内，应规划好接口部分标识符(变量、结构、函数及常量等)的命名，防止编译、连接时产生冲突。对接口部分的标识符应该有更严格的限制，防止冲突。例如，可规定接口部分的变量与常量之前加上“模块”标识。

(10) 用正确的反义词组命名具有互斥意义的变量或具有相反动作的函数。

(11) 除了编译开关、头文件等特殊应用外，应避免使用_EXAMPLE_TEST_之类以下划线开始和结尾的定义。

(12) 程序中不要出现标识符完全相同的局部变量和全局变量，尽管二者的作用域不同而不会发生冲突，但容易使人误解。

7.5.3 语句构造与程序书写

一个程序如果写得密密麻麻，分不出层次结构，是很难读懂的。优秀的程序结构及规范的书写格式能增强程序可读性，减少错误发生，提高可维护性。

1. 语句构造

语句是构成程序的基本单位，语句的构造方式和书写格式对程序的可读性具有非常重要的决定作用。语句构造是根据设计时确定的软件逻辑结构来编写语句，需要注意以下问题：

(1) 语句力求简单、直接，不要因为片面地追求效率而使语句复杂化。

(2) 使用括号来清晰地表示算术表达式和逻辑表达式。

(3) 由于人的一般思维方式对逻辑非运算不太适应，因此在条件表达式中应尽量不使用否定的逻辑表示。

(4) 尽量采用结构化设计的三种基本控制结构编写程序。

(5) 为了不破坏结构化程序设计中结构的清晰性，在程序中尽量不使用强制转移语句Goto。

(6) 避免不必要的转移。

(7) 为了便于理解程序，避免过多的循环嵌套和条件嵌套。

(8) 为了缩短程序代码，在程序中应尽可能地使用编译系统提供的标准函数。

(9) 对于程序中需要重复出现的代码段，应将其用独立模块(函数或过程)实现。

(10) 避免采用过于复杂的条件测试。

(11) 避免从循环引出多个出口。

2. 程序书写

为了便于人们阅读程序(特别是大型程序)，清晰整齐的书写格式是必不可少的。以下列出了程序书写时需注意的几个主要问题：

(1) 虽然许多语言都允许在一行书写多个语句，但为了程序看起来更加清楚，最好在一行只书写一条语句。

(2) 书写语句时，应该采用递缩格式，使程序层次更加清晰。

(3) 较长的语句(>80 字符)要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

(4) 对两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符(如－>)，后不应加空格。

(5) 在模块之间通过加入空行进行分隔。

(6) 为了便于区分程序中的注释，最好在注释段的周围加上边框。

(7) 避免使用空的 Else 语句和 If…Then If…语句，这种结构容易使人产生误解。

3. 其他注意事项

(1) 在不影响功能与性能时，做到结构清晰第一、效率第二。

(2) 不能刻意追求技巧,力求简洁、直截了当。
(3) 先保证程序正确,再设法提高速度。
(4) 建立公共过程时应确保具有独立的功能。
(5) 利用信息隐藏,确保各个模块的独立性。
(6) 要模块化,模块功能尽可能单一化,模块间的耦合清晰可见。
(7) 确保所有变量使用前都进行初始化。
(8) 不要用浮点数做相等比较。
(9) 大程序应分块编写,测试、调试后再集成。
(10) 不要修补不好的程序,要重新编写,也不要一味地追求代码复用。
(11) 通过代码走读及审查方式对代码进行检查。
(12) 应遵循标准。

7.6 软件调试

软件调试就是通过调试手段解决软件中的问题。

1. 软件调试的步骤

软件调试的任务是确定并改正程序中的潜在错误。调试活动的执行步骤如下:
(1) 从错误的外部表现形式入手,确定程序中的出错位置。
(2) 研究有关部分程序,找出错误的内在原因。
(3) 修改设计和代码,以排除这个错误。
(4) 重复进行暴露了这个错误的原始测试或某些有关测试,以确保该错误已经消除且未引入新的错误。
(5) 如果所做的修改无效,则撤销这次修改,重复上述过程,直至找到一个有效的解决办法为止。

2. 查找错误的难点

从技术角度看,查找错误的难点在于:
(1) 现象与原因所处的位置可能相距甚远。也就是说,现象可能出现在程序的一个位置,而原因可能在离此很远的另一个位置。在高耦合的程序结构中这种情况更为明显。
(2) 当其他错误得到纠正时,某一错误表现出的现象可能暂时消失,但并未真正排除。
(3) 现象实际并非是由错误原因(例如舍入不精确)引起的。
(4) 现象可能是由一些不容易发现的人为错误引起的。
(5) 错误是由时序问题引起的,与处理过程无关。
(6) 现象是由难以精确再现的输入状态引起的。
(7) 现象可能是周期出现的。

3. 软件调试的方法

调试的关键在于推断程序内部错误的位置及其原因。为此,可以采用强行排错、回溯法

排错、归纳法排错和演绎法排错四种软件调试方法。

1）强行排错

强行排错是目前使用较多但效率较低的调试方法。常用的方法有以下几种：

（1）通过内存全部打印来排错。这种方法是将计算机存储器和寄存器的全部内容打印出来，然后在这些大量的数据中寻找出错位置。虽然有时可以获得成功，但效率极低。这种方法具有以下缺点：

① 建立内存地址与源程序变量之间的对应关系很困难。仅汇编和手编程序才有可能。

② 人们将面对大量八进制或十六进制数据，其中大多数与所查错误无关。

③ 一个内存全部内容打印清单只显示了源程序在某一瞬间的状态，即静态映像。但为了发现错误，需要程序随时间变化的动态过程。

④ 一个内存全部内容打印清单不能反映在出错位置处程序的状态。程序在出错时刻与打印信息时刻之间的时间间隔内所做的事情可能会掩盖所需要的线索。

⑤ 缺乏从分析全部内存打印信息来找到错误原因的算法。

（2）在程序特定位置设置打印语句。这种方法是把打印语句插在出错源程序的各个关键变量改变部位、重要分支部位和子程序调用部位，跟踪程序执行，监视重要变量的变化。这种方法能显示出程序的动态过程，允许人们检查与源程序有关的信息。因此，比全部打印内存信息优越，但是也有以下缺点：

① 可能输出大量需要分析的信息，大型程序或系统更是如此，造成费用过大。

② 必须修改源程序以插入打印语句，这种修改可能会掩盖错误，改变关键的时间关系或把新的错误引入程序。

（3）使用自动调试工具。这种方法是利用某些程序语言的调试功能，或专门的交互式调试工具，分析程序的动态过程，而不必修改程序。可供利用的典型的语言功能有打印出语句执行的追踪信息、追踪子程序调用以及指定变量的变化情况。自动调试工具的功能是设置断点，当程序执行到某个特定语句或某个特定的变量值改变时，程序暂停执行。程序员可以在终端上观察程序此时的状态。

在应用以上任何一种方法之前，都应对错误征兆进行全面彻底的分析，得出对出错位置及错误性质的推测，再使用一种适当的排错方法来检验推测的正确性。

2）回溯法排错

回溯法排错是一种在小程序中常用的有效排错方法。一旦发现了错误，先分析错误征兆，确定最先发生“症状”的位置，然后人工沿程序的控制流程向回追踪源程序代码，直至找到错误根源或确定错误产生的范围。

回溯法排错对于小型程序很有效，往往能把错误范围缩小到程序中的一小段代码，仔细分析这段代码，便不难确定出错的准确位置。但对于大型程序，由于回溯的路径数目较多，回溯会变得很困难。

3）归纳法排错

归纳法排错是一种从特殊推断一般的系统化思考方法。归纳法排错的基本思想是从一些线索（错误征兆）着手，通过分析它们之间的关系找出错误。归纳法排错的步骤如图 7.8 所示。

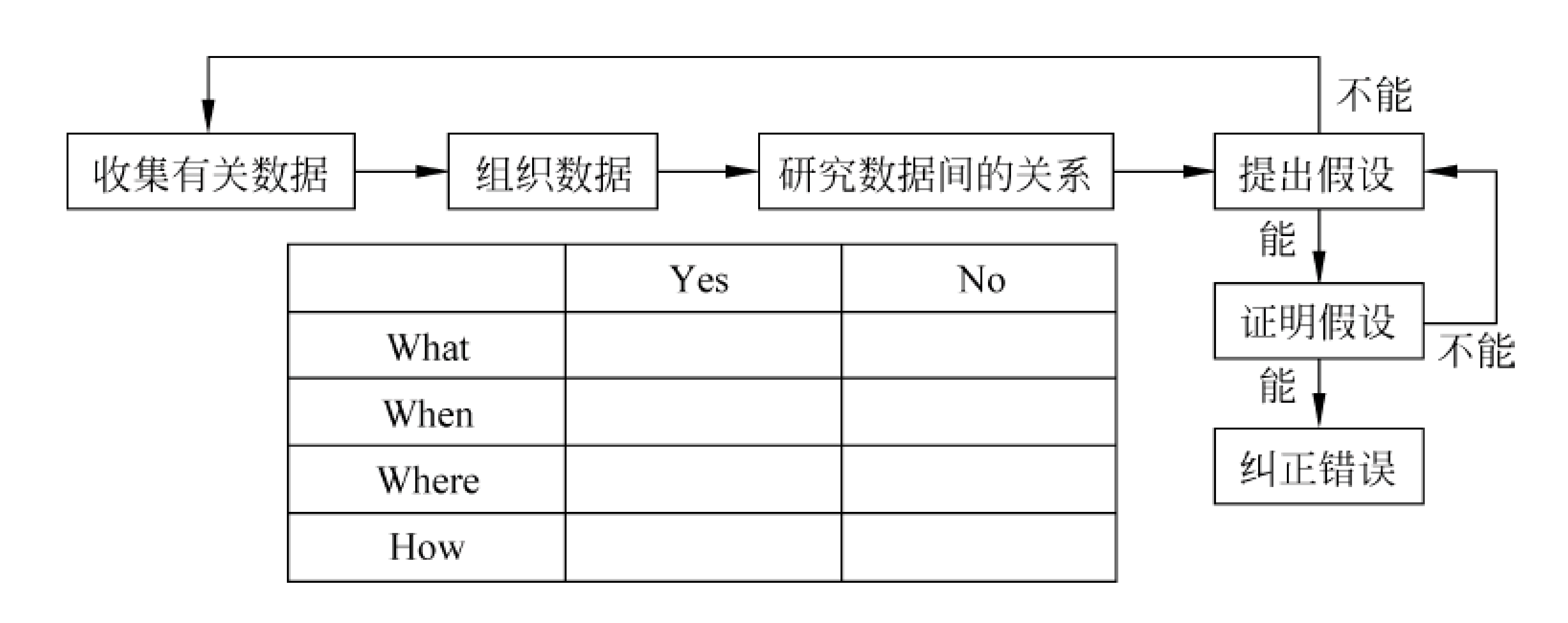


图 7.8 归纳法排错的步骤

(1) 收集有关数据。列出所有已知的测试用例和程序执行结果,看哪些输入数据的运行结果是正确的,哪些输入数据的运行结果是错误的。

(2) 组织数据。由于归纳法是从特殊到一般的推断过程,所以需要组织整理数据,以便发现规律。构造线索的常用技术是"分类法"。用图 7.8 中所示的 3W1H 形式来组织可用的数据。其中,"What"列出一般现象;"Where"说明发生现象的地点;"When"列出现象发生时所有已知情况;"How"说明现象的范围和量级;而在"Yes"和"No"两列中,"Yes"描述了出现错误现象的 3W1H,"No"作为比较,描述了没有错误现象的 3W1H。通过分析,找出矛盾。

(3) 提出假设。分析线索之间的关系,利用在线索结构中观察到的矛盾现象,设计一个或多个关于出错原因的假设。如果一个假设也提不出来,归纳过程就需要收集更多的数据。此时,应当再设计与执行一些测试用例,以获得更多的数据。如果提出了许多假设,则首先选用最有可能成为出错原因的假设。

(4) 证明假设。把假设与原始线索或数据进行比较,若能完全解释一切现象,则假设得到证明;否则,认为假设不合理,或不完全,或是存在多个错误,以致只能消除部分错误。有人想越过这一步,立刻就去改正错误,这样,假设是否合理、是否完全、是否同时存在多个错误都不太清楚,因此就不能有效地消除多个错误。

4) 演绎法排错

演绎法排错是一种从一般原理或前提出发,经过排除和精化过程推导出结论的思考方法。演绎法排错是测试人员首先根据已有的测试用例,设想并枚举出所有可能出错的原因作为假设,然后再从原始测试数据或新的测试中逐个排除不可能正确的假设,最后再用测试数据验证余下的假设是出错的原因。演绎法排错的步骤如图 7.9 所示。

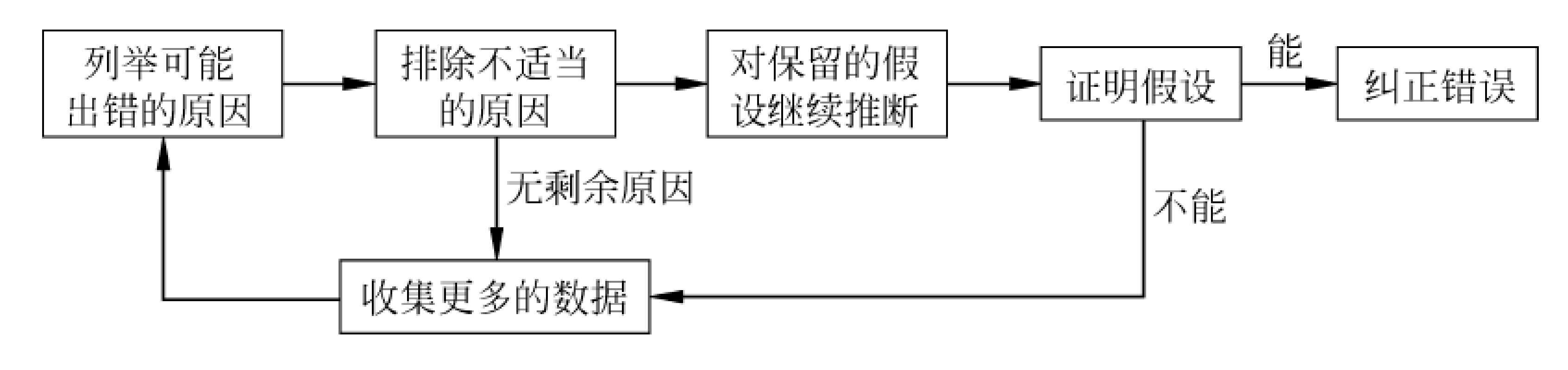


图 7.9 演绎法排错的步骤

(1) 列举可能出错的原因。列出所有可能错误的原因,不需要完全解释,而仅仅是一些可能因素的假设。通过出错原因,可以组织、分析现有数据。

(2) 排除不适当的原因。仔细分析已有的测试数据，寻找矛盾，力求排除前一步列出的所有原因。如果所有原因都被排除了，则需要补充一些数据(测试用例)，以建立新的假设；如果保留下来的假设多于一个，则选择可能性最大的原因作为基本假设。

(3) 对保留的假设继续推断。利用已知线索，进一步改进余下的假设，使之更加具体化，以便可以精确地确定出错位置。

(4) 证明假设。把假设与原始线索或数据进行比较，若能完全解释一切现象，则假设得到证明；否则，认为假设不合理，或不完全，或是存在多个错误，以致只能消除部分错误。有人想越过这一步，立刻就去改正错误，这样，假设是否合理、是否完全、是否同时存在多个错误都不甚清楚，因此就不能有效地消除多个错误。

7.7 程序效率

程序效率是指程序的执行速度及程序占用的内存空间，分为全局效率、局部效率、时间效率及空间效率。全局效率是从整个系统角度考虑的效率；局部效率是从模块或函数角度考虑的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，包括机器代码空间大小、数据空间大小、堆栈空间大小等。

1. 提高效率的准则

提高效率的准则有以下几条：

(1) 效率是一个性能要求，应当以用户需求为准，在需求分析阶段给出，但不要单纯地追求高效率。应在保证软件系统的正确性、稳定性、可读性及可测性的前提下提高效率。

(2) 局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。

(3) 好的设计可以提高效率。程序效率与程序的简单性相关，一般情况下是程序越简单效率越高。

(4) 任何对效率无重大改善，且对程序的简单性、可读性和正确性不利的程序设计方法都是不可取的。

(5) 通过对系统的数据结构划分与组织改进，以及对程序算法的优化来提高空间效率，这是解决软件空间效率的根本办法。

(6) 在优化程序效率时，应当找出限制效率的“瓶颈”，不要在无关紧要处进行过多的优化。

(7) 有时候时间效率和空间效率是对立的，要具体分析哪个更重要而做出优化选择。

2. 算法对效率的影响

源程序的效率与详细设计阶段确定的算法的效率直接有关。把详细设计转换成源程序代码后，算法效率反映为程序的执行速度和存储容量的要求。转换过程的指导原则是：

(1) 在编写程序前，尽可能化简有关的算术表达式和逻辑表达式。

(2) 仔细检查算法中的嵌套循环，尽可能将某些语句或表达式移到循环体外面。

(3) 尽量避免使用多维数组。

(4) 尽量避免使用指针和复杂的表达式。

(5) 采用“快速”的算术运算。

(6) 不要混淆数据类型,避免在表达式中出现类型混杂。

(7) 尽量采用整数算术表达式和布尔表达式。

(8) 选用高效率算法。

许多编译程序具有“优化”功能,能够自动生成高效率的目标代码。可以通过剔除重复的表达式计算,采用循环求值法、快速的算术运算,以及采用一些能够提高目标代码运行效率的算法来提高效率。对于效率至上的应用来说,这样的编译程序是很有效的。

3. 影响存储效率的因素

在大中型计算机系统中,存储限制不再是主要问题。在这种环境下,对内存采取基于操作系统分页功能的虚拟存储管理,给软件提供了巨大的逻辑地址空间。这时,存储效率与操作系统的分页功能直接有关,并不是指要使所使用的存储空间达到最小。

采用结构化程序设计,将程序功能合理分块,使每个模块或一组密切相关模块的程序体积大小与每页的容量相匹配,可减少页面调度,减少内外存交换,提高存储效率。

在微型计算机系统中,存储容量对软件设计和编码的制约很大。因此要选择可生成较短目标代码且存储压缩性能优良的编译程序,有时需要采用汇编程序。通过程序员富有创造性的努力,提高软件时间与空间效率。

提高存储效率的关键是程序的简单性。

4. 影响输入输出的因素

输入输出可分为两种类型:一种是面向人(操作员)的输入输出;另一种是面向设备的输入输出。如果操作员能够十分方便、简单地输入数据,或者能够十分直观、一目了然地了解输出信息,则可以说面向人的输入输出是高效的。至于面向设备的输入输出,分析起来比较复杂。从详细设计和程序编码的角度来说,可以提出一些提高输入输出效率的指导原则:

(1) 输入输出的请求应当最小化。

(2) 对于所有的输入输出操作,安排适当的缓冲区,以减少频繁的信息交换。

(3) 对辅助存储(例如磁盘),选择尽可能简单的、可接受的存取方法。

(4) 对辅助存储的输入输出,应当成块传送。

(5) 对终端或打印机的输入输出,应考虑设备特性,多采用并发运行,改善输入输出的质量和速度。

(6) 任何不易理解的、对改善输入输出效果关系不大的措施都是不可取的。

(7) 任何不易理解的、所谓“超高效”的输入输出都是毫无价值的。

(8) 良好的输入输出程序设计风格对提高输入输出效率具有明显的效果。

7.8 程序安全性

提高软件质量和可靠性的技术大致可分为两类:一类是避开错误技术,即在开发过程中不让错误潜入软件的技术;另一类是容错技术,即对某些无法避开的错误,使其影响减至最小的技术。避开错误技术是进行质量管理、保证产品质量必不可少的技术,也就是软件工

程中研究的先进的软件分析技术、开发技术和管理技术。

无论使用多么高明的避开错误技术都无法做到百分之百的完美和绝对无错误，仍需要采用容错技术。实现容错的主要手段是冗余程序设计和防错程序设计。

7.8.1 冗余程序设计

冗余是改善系统可靠性的一种重要技术。在硬件系统中，采用冗余技术是指提供额外的元件或系统，使其与主系统并行工作。这时有两种情况：一种情况是让连接的所有元件都并行工作，当有一个元件出现故障时就退出系统，而由冗余元件接续它的工作，维持系统运转，有时将这种结构称为自动重组结构；另一种情况是系统最初运行时，由原始元件工作，当该元件发生故障时，由检测线路(有时由人工完成)把备用元件接上(或把开关拨向备用元件)，使系统继续运转。第一种情况称为并行冗余，也称热备用或主动冗余；第二种情况称为备用冗余，也称冷备用或被动冗余。

在软件系统中，采用冗余技术是指解决一个问题必须设计出两个不同的程序，包括采用不同的算法和设计，而且编程人员也应该不同。

7.8.2 防错程序设计

防错程序设计可分为主动式和被动式两种。

1. 主动式防错程序设计

主动式防错程序设计是指周期性地对整个程序或数据库进行搜查，或在空闲时搜查异常情况。主动式防错程序设计既可在处理输入信息期间使用，也可在系统空闲时间或等待下一个输入时使用。以下列出的检查均适合于主动式防错程序设计：

- 内存检查；
- 标志检查；
- 反向检查；
- 状态检查；
- 连接检查；
- 时间检查；
- 其他检查。

2. 被动式防错程序设计

被动式防错程序设计是指必须等到某个输入之后才能进行检查，也就是说达到检查点时，才能对程序的某些部分进行检查。被动式防错程序设计中要检查的项目如下：

- 来自外部设备的输入数据，包括范围、属性是否正确；
- 由其他程序提供的数据是否正确；
- 数据库中的数据，包括数组、文件、结构、记录是否正确；
- 操作员的输入，包括输入的性质、顺序是否正确；
- 堆栈的深度是否正确；

- 数组界限是否正确；
- 表达式中是否出现零分母情况；
- 正在运行的程序版本是否是所期望的；
- 通过其他程序或外部设备的输出数据是否正确。

思考题

1. 软件实现主要完成哪些工作？
2. 输入设计的原则是什么？输入设计有哪些具体方式？
3. 设计原始单据的原则是什么？
4. 常用的数据校验方法有哪些？
5. 输出设计的内容包括哪些？
6. 理解报表模块原理。
7. 常用的图形输出类型有哪几种？各有何特点？
8. 简述屏幕界面设计的八条“黄金规则”。
9. 屏幕界面设计内容主要包括哪些方面？
10. 屏幕布局应遵循哪些原则？
11. 从软件工程的角度来看，根据程序设计语言的发展历程，程序设计语言如何分类？
12. 选择程序设计语言应遵循怎样的标准？
13. 简述编程风格的重要性。
14. 编写程序时应遵循怎样的风格？
15. 标识符命名的主要规则是什么？
16. 构造语句时应注意哪些问题？
17. 书写程序时需注意哪些问题？
18. 软件调试有哪些具体方法？
19. 提高程序效率的准则是什么？
20. 如何理解冗余程序设计？
21. 防错程序设计可分为哪两种？

第8章 软件测试

软件测试是为了发现错误而执行程序的过程。软件测试不仅是软件开发阶段的组成部分，而且在整个软件工程过程(即软件定义、设计和实现等阶段)中具有非常重要的作用。软件测试是软件质量保证的关键环节，直接影响着软件的质量评估。软件测试不仅要讲究策略，更要讲究时效性。验收测试作为软件测试过程的最后一个环节，对软件质量、软件的可交付性和软件项目的实施周期起到“一锤定音”的作用。

8.1 软件测试概述

8.1.1 软件测试过程

软件测试过程按测试的先后顺序可分为单元测试、集成测试、确认测试、系统测试，分别与软件开发过程的软件编码、软件设计、软件需求、系统需求(整个项目的需求)相对应。软件测试过程流程如图 8.1 所示。

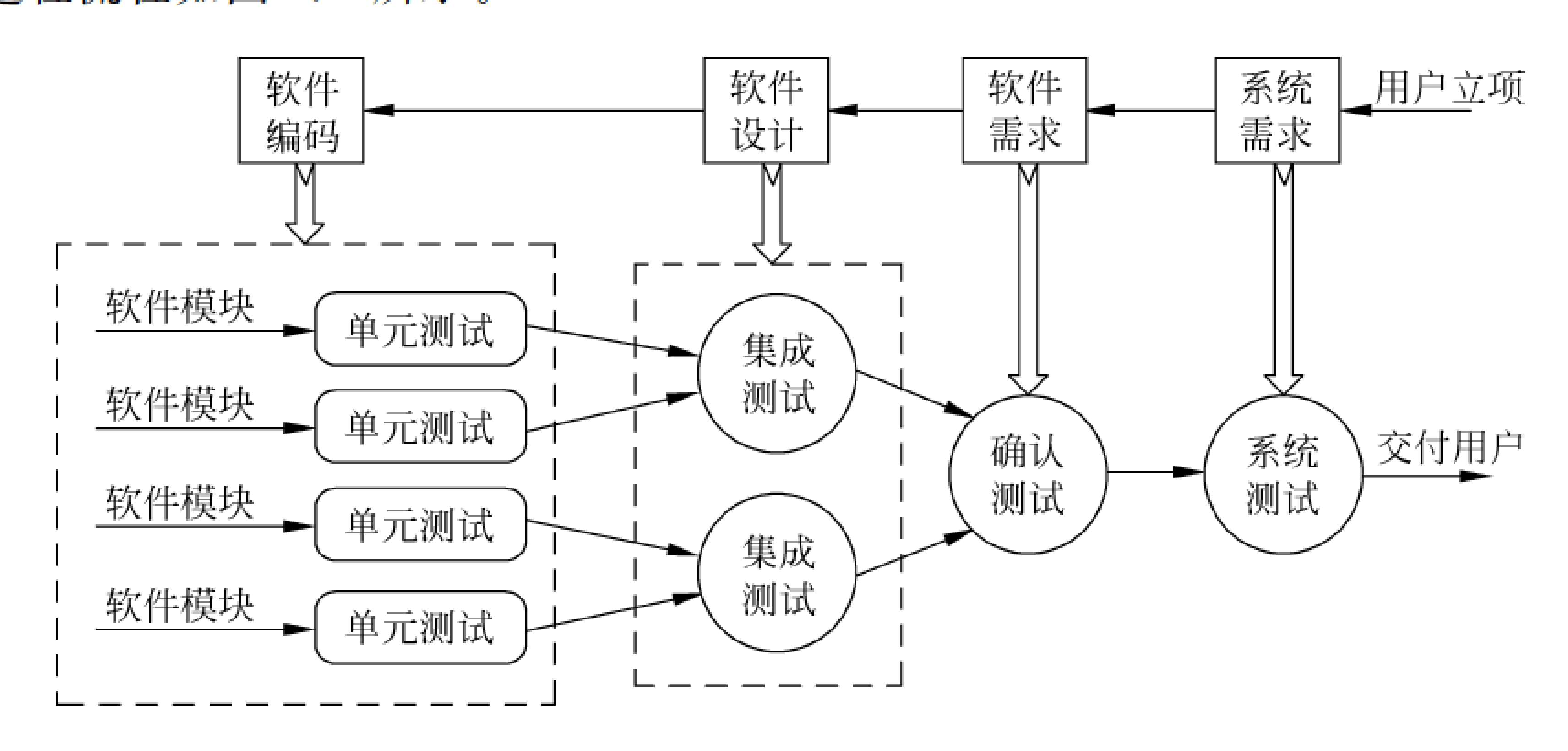


图 8.1 软件测试过程流程

(1) 单元测试。对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能，并发现程序内部错误。

(2) 集成测试。把已测试过的模块组装起来，主要对与设计相关的软件体系结构的构造进行测试。

(3) 确认测试。检查已实现的软件是否满足需求规格说明中确定的各种需求，以及软

件配置是否完全、正确。

（4）系统测试。把经过确认的软件纳入实际运行环境中，与其他系统成分组合在一起进行测试。

8.1.2 软件测试原则

软件测试从不同角度出发会派生出两种不同的测试原则。从用户角度出发，希望通过软件测试充分暴露软件中存在的问题和缺陷，从而考虑是否可以接受该产品；从开发者角度出发，希望通过测试表明软件产品不存在错误，已经正确地实现了用户的需求，确立人们对软件质量的信心。无论从哪个角度出发，都必须做好软件测试工作，都应掌握一定的策略和方法，通常遵循以下一些原则：

（1）尽早和不断地进行软件测试。不应把软件测试仅仅看做软件开发的一个独立阶段，而要贯穿到软件开发的各个阶段中。坚持在软件开发的各个阶段实施技术评审，才能在开发过程中尽早发现和预防错误。随着开发过程的进行，发现和修复缺陷的平均代价呈指数级增长。

（2）开发人员原则上不测试自己的程序。程序员应避免测试自己编写的程序，开发小组也应尽可能避免测试自己小组开发的程序。如果条件允许，最好建立独立的软件测试机构。这一点不能与程序调试相混淆。

（3）完全测试程序是不可能的。初涉软件测试者认为可以对软件进行完全测试，找出所有的软件缺陷，并使软件臻于完美。这种想法固然很好，但在测试过程中是不现实的。即使最简单的程序也未必可行，因为输入量太多，软件实现途径太多。

（4）软件测试是有一定风险的行为。如果不测试所有情况，那就是选择了风险。有可能程序员恰好留下一个软件缺陷，测试员没有测试，用户却会用到它，并发现缺陷，这就成为修复代价很高的缺陷。软件测试员要掌握如何把软件缺陷减少到可以控制的范围，并针对软件缺陷可能带给用户的风险进行分析，寻找最合适的测试用例。

（5）充分注意测试中的群集现象。在被测试程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集现象也被许多程序测试的实践所证实。根据这个规律，应当对错误群集的程序段进行重点测试。

（6）严格执行测试计划，排除测试的随意性。测试之前应仔细研究被测试的项目，对每一项测试做出周密的计划，包括被测试程序的功能、输入输出、测试内容、进度安排、资源需求、测试用例选择、控制方式和过程等；还包括系统的组装方式、跟踪规程、调试规程、回归测试的规定以及评价标准等。对于测试计划，要明确制定，不能随意修改。

（7）应当对每一个测试结果进行全面检查。有些错误征兆在输出实例结果时已明显地表现出来，但是如果不仔细全面地检查测试结果，就会使这些错误被遗漏掉。所以必须对预期的输出结果明确定义，对结果仔细分析检查，抓住征兆，发现错误。

（8）妥善保存测试文档。妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。

（9）软件测试的 Pareto 法则（8：2）。20％的模块产生 80％的缺陷；20％的缺陷消耗 80％的维护费用；20％的原因导致了 80％的缺陷。

（10）并非所有的软件缺陷都能被修复。在软件测试过程中不能修复所有缺陷并不意

味着软件测试人员未达到目的,或者项目小组将发布质量欠佳的产品。项目小组需要对每一个缺陷进行取舍,根据风险决定哪些需要修复,哪些不需要修复。

不需要修复所有缺陷的原因如下:

① 没有足够的时间。通常是软件功能多或者交付产品时间短,代码编写人员和软件测试人员少,而且在项目进度中没有为软件测试和修改缺陷留出足够的时间。

② 修复风险太大。软件复杂,难以理清头绪。修复一个软件缺陷可能导致其他缺陷出现。在紧迫的产品发布进度压力下,修改软件将冒很大的风险。

③ 不值得修复。不常出现的缺陷或不常用功能中出现的缺陷,用户可能有办法预防或避免,通常不用修复。

(11) Bug 的 80%原则。一般情况下,在分析、设计、实现阶段的复审和测试工作能够发现 80%的 Bug,而系统测试能够找出其余 Bug 中的 80%,最后约 5%的 Bug 只有在用户长期的使用过程中才能发现。因此测试要尽可能多地发现错误,而并非是发现所有错误。

(12) 软件测试人员在项目小组中的处事原则。软件测试人员的任务是检查和指正同事的工作、挑毛病、公布发现的问题,这项工作是不受欢迎的。测试人员与开发人员和睦相处的建议如下:

① 尽早找出软件缺陷。

② 不要总报告坏消息。

③ 控制情绪。软件测试人员发现严重的软件缺陷时,如果兴冲冲地告诉开发人员,会令人反感。可以通过电子邮件或管理软件的方法通知开发人员。

8.2 软件测试方法

软件测试方法很多,常用的有静态测试、动态测试、覆盖分析、黑盒测试、白盒测试等。

8.2.1 静态测试与动态测试

根据测试时是否实际运行程序,可以将测试分为静态测试和动态测试。

1. 静态测试

静态测试不实际执行程序代码,主要对软件的编程格式、结构等方面进行评估并发现可能存在的错误。静态测试具有以下优点:

(1) 不必动态地运行程序,不必设计测试用例,不用判读结果。

(2) 可以由人工进行,充分发挥人的逻辑思维优势,检测出错误的水平很高。

(3) 不需要特别条件,容易开展。

静态测试主要由代码检查和静态分析组成。

1) 代码检查

代码检查包括代码审查、代码走查、桌面检查等。

(1) 代码审查(code inspection)是由若干个程序员与测试员组成一个审查小组,集体阅读并讨论程序,对照代码审查单检查程序代码,以发现程序中的缺陷、违反标准之处和其他

问题。

(2) 代码走查(code walkthrough)是由若干个程序员与测试员组成一个走查小组,集体阅读并讨论程序,设计一些典型的测试用例,用"人脑"执行并检查程序,以找出程序中的缺陷。

(3) 桌面检查(desk checking)是由程序员阅读自己编写的程序,以发现程序中的缺陷。这种方法有下述三个方面的缺点:一是由于心理上的原因,容易对自己的程序存在偏爱,没有发现错误的欲望;二是由于人的思维定式,有些习惯性的错误自己不易发现;三是如果对功能的理解存在错误,那么只靠自己是不容易纠正的。所以这种方法效率不高,但可以作为一种自我检查程序中存在的明显疏漏或笔误的方法。

代码审查和代码走查不仅比桌面检查优越得多,而且与计算机的测试方法相比也有许多优点。一旦发现错误,就能知道错误的性质和位置,因而程序调试花费的代价就低;一次能揭示一批错误,而不是一次只揭示一个错误。如果使用计算机测试,通常仅能揭示错误的征兆(例如程序不能正常运行),而对错误的性质与位置还要逐个查找。

研究表明,代码审查和代码走查发现某类错误比使用计算机的测试方法更为有效,而对于另一类错误,情况则正好相反。由此可见,代码审查和代码走查方法与使用计算机的测试方法是相互补充的,缺少任何一种方法都会使错误的检测率下降。

2) 静态分析

静态分析是分析一个程序,但并不实际执行这个程序,可分为手工和自动两类。静态分析的技术结构如图 8.2 所示。

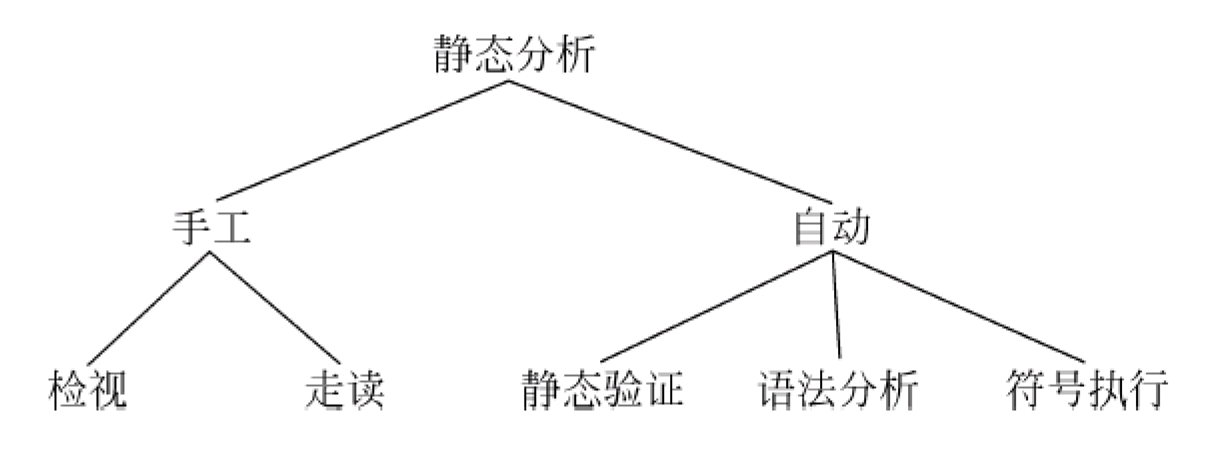


图 8.2　静态分析的技术结构

手工静态分析包括检视和走读两种方法。检视是使用预先定义好的检视规则,对代码、文档等工作产品进行检查。实际操作中,检视过程一般根据检查表来进行。走读又称走查,是由分析者对程序的运行细节进行想象,以检查程序的正确性,是一个类似同行评审的过程。

自动静态分析包括静态验证、语法分析、符号执行等,需要相应的工具。自动静态分析用于发现代码、文档等工作产品在规范化和形式化方面的缺陷很方便。如果要通过自动静态分析发现更深层次的问题,则在形式化和规范化方面要求很高。有些软件开发工作在文档规范化和形式化方面很差,使用这些工具和技术就非常困难。

2. 动态测试

动态测试方法是通过运行软件来检查软件的动态行为和运行结果是否正确的方法。动态测试具有以下特点:

(1) 实际运行被测程序,取得程序运行的真实情况和动态情况,再进行分析。

(2) 必须生成测试数据来运行程序,测试质量依赖于测试数据。

(3) 生成测试数据、分析测试结果的工作量很大,开展测试工作费时、费力。

(4) 动态测试中涉及多方面的工作,人员多、设备多、数据多,要求有较好的管理和工作规程。

动态测试包括以下几种:

(1) 功能确认与接口测试。功能确认与接口测试包括各个单元功能的正确执行以及单元接口、局部数据结构、重要的执行路径、错误处理路径和影响上述几个点的边界条件等内容。

(2) 覆盖率分析。覆盖率分析主要是对代码的执行路径覆盖范围进行评估,语句覆盖、判定覆盖、条件覆盖、条件/判定覆盖、修正条件/判定覆盖、基本路径覆盖等都是从不同要求出发,为设计测试用例提供依据。

(3) 性能分析。程序运行缓慢是开发过程中常见的问题。如果不能解决应用程序的性能问题,将降低并影响程序的质量,查找和修改性能瓶颈成为调整代码性能的关键。目前,性能分析工具大致分为纯软件的测试工具、纯硬件的测试工具、软硬件结合的测试工具三类。

(4) 内存分析。内存泄露会导致系统运行崩溃,尤其对于嵌入式系统这种资源比较匮乏、使用非常广泛且往往又处于重要部位的软件,将可能导致无法预料的损失。通过测量内存使用情况,可以了解程序内存分配的真实情况,发现对内存的不正常使用,在问题出现前发现征兆,在系统崩溃前发现内存泄露错误;通过内存分析,精确显示发生错误时的上下文情况,指出发生错误的原因。

8.2.2 黑盒测试与白盒测试

黑盒测试和白盒测试是两种不同的测试方法,有的测试阶段是以黑盒测试为主,有的测试阶段是以白盒测试为主,有的测试阶段则将二者结合起来使用。

1. 黑盒测试

黑盒测试也称功能测试或数据驱动测试,是已知产品所应具有的功能,通过测试来检测每个功能是否都正确。在测试时,把程序看成是一个不能打开的黑盒子,在完全不考虑程序内部结构和内部特性的前提下,测试者对程序接口进行测试,只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能接收输入数据而产生正确的输出信息,并且保持外部信息(如数据库或文件)的完整性。黑盒测试原理如图 8.3 所示。



图 8.3 黑盒测试原理

黑盒测试主要有等价类划分、边界值分析、因果图、决策表等测试方法,主要用于软件确认测试。黑盒测试着眼于程序外部结构,不考虑内部逻辑结构,针对软件界面和软件功能进行测试。黑盒测试是穷举输入测试,只有把所有可能的输入都作为测试情况使用,才能找出程序中的所有错误。实际上测试情况有无穷多个,不仅要测试所有合法的输入,而且还要对不合法但是可能的输入进行测试。

2. 白盒测试

白盒测试也称结构测试或逻辑驱动测试,是一种按程序内部的逻辑结构和编码结构设

计测试数据的测试方法。已经明确产品内部工作过程,通过测试来检测产品内部动作是否按照规格说明书进行,按照程序内部结构测试程序,检验程序中的每条通路是否都能按预定要求正确地工作,而不管软件功能。白盒测试主要有独立路径测试、逻辑判断测试、数据结构测试、覆盖率测试等方法,主要用于软件验证。白盒测试原理如图 8.4 所示。

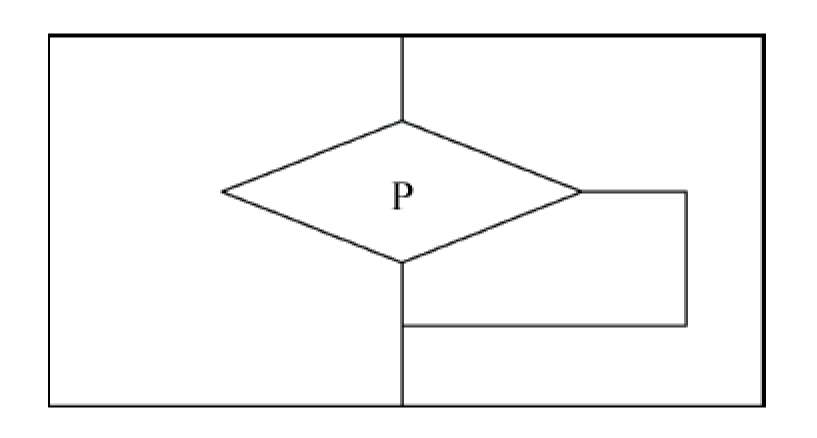


图 8.4 白盒测试原理

白盒测试应全面了解程序内部的逻辑结构,对所有逻辑路径进行测试。白盒测试是穷举路径测试。测试时,测试者必须检查程序的内部结构,从检查程序的逻辑着手,生成测试数据。但是贯穿程序的独立路径数是天文数字,即使每条路径都测试了仍然可能有错误。原因是:第一,穷举路径测试不能查出程序违反了设计规范,即程序本身就是错误的;第二,穷举路径测试不可能查出程序中因遗漏路径而产生的错误;第三,穷举路径测试可能无法发现一些与数据相关的错误。

3. 黑盒测试与白盒测试的比较

无论黑盒测试还是白盒测试,都不可能对软件进行完整而彻底的测试。黑盒测试从考虑输入数据的角度出发验证软件功能,除非输入数据穷举,否则不可能进行完全测试。白盒测试从程序结构出发,由于程序结构的复杂性,路径数本身有时是不能确定的,即使确定下来也往往是天文数字,所以要测试程序的全部结构(每一路径)也是不现实的。另一方面,黑盒测试基于需求规格说明,如果需求规格说明存在错误,使用黑盒测试方法也无法发现这类错误。白盒测试则基于程序的逻辑结构,如果逻辑存在错误或者遗漏,使用白盒测试方法也无法发现这类错误。两种测试方法的比较如表 8.1 所示。

表 8.1 黑盒测试与白盒测试的比较

比较项目	黑 盒 测 试	白 盒 测 试
主要应用	功能测试	结构测试
特点	• 不基于对系统内部的设计和实现; • 用例设计基于功能的定义和需求说明书; • 关注于测试数据选择和测试结果分析	• 需要了解系统的整体设计和实现; • 对源代码进行审查; • 在单元测试阶段发现大量缺陷; • 关注于系统的控制流和数据流
优点	• 能确保从用户的角度出发进行测试	• 能对程序内部的特定部位进行覆盖测试
缺点	• 对用例设计人员的经验要求较高,包括数据选择、对潜在错误的敏感性等; • 对于内部实现的 Bug 不容易发现; • 不能提供直观的测试覆盖率	• 不能确保系统是否完全符合需求; • 白盒测试的代价会大于黑盒测试; • 需要源代码完成后才能进行测试
主要方法	• 等价类划分; • 边界值分析; • 因果图; • 决策表测试	• 独立路径测试; • 逻辑判断测试; • 数据结构测试; • 覆盖率测试

4. 灰盒测试

灰盒测试是介于黑盒测试和白盒测试之间的测试，关注输出对于输入的正确性，同时也关注内部表现。但关注程度不像白盒测试那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，有时候输出是正确的，但内部却是错误的，这种情况非常多，如果每次都通过白盒测试来操作，效率很低，因此需要采取灰盒测试方法。

灰盒测试结合了白盒测试和黑盒测试要素，考虑了用户端、特定的系统知识和操作环境，并在系统组件的协同性环境中评价应用软件设计。

灰盒测试涉及输入和输出，但是关于代码和程序操作等通常在测试人员视野之外的信息设计阶段测试。

8.3 测试用例设计技术

测试用例是针对特定测试对象而开发的一组输入、预置条件和预期结果。一个测试用例可以理解为一组数据。通常相对于被测试对象，需要用普通数据、边界（极限）数据和错误数据来设计测试用例，以便测试各种可能情况。

只有设计好的测试用例，才能通过采用尽量少的用例数量获得较大的测试覆盖率，提高测试质量。从广义上讲，测试用例可分为两类：黑盒测试用例和白盒测试用例。

8.3.1 黑盒测试用例设计

黑盒测试属于穷举输入测试，只有将所有可能的输入都作为测试情况来使用，才能查出程序中的所有错误，但通常把所有可能的输入都测试又不可能，所以测试用例设计有很多技巧。

黑盒测试着眼于程序的外部结构，主要针对软件界面、软件功能、外部数据库访问、软件初始化等方面设计测试用例。不同的黑盒测试技术对应不同的测试用例设计技术。

1. 等价类划分

在完全不考虑程序内部结构的情况下，只根据程序的规格说明书设计测试用例，把程序的输入范围划分成若干部分，然后从每一部分中选取少量代表性数据作为测试用例。

1）划分等价类

首先把数据极多的输入数据（有效的和无效的）划分为若干类。所谓等价类，是指某个输入域的子集合。在该子集合中，各个输入数据对于发现程序中的错误是等效的。因此，把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入，就可以用少量代表性测试数据取得较好的测试效果。

有效等价类是指对于程序规格说明书来说，由合理的、有意义的输入数据构成，用于检查需求规格说明书规定的程序的功能和性能。

无效等价类是指对于程序规格说明书来说，由不合理的、无意义的输入数据构成，用于检查程序中功能和性能的实现是否有不符合规格说明书的内容。

2）等价类划分的原则

（1）如果输入条件规定了取值范围或值的个数，则可以确定一个有效等价类和两个无效等价类。例如，$1<x<99$ 是一个有效等价类，$x\geq 99$ 和 $x\leq 1$ 是两个无效等价类。

（2）如果输入条件规定了输入值的集合，或者是规定了"必须如何"的条件，就可以确定一个有效等价类和一个无效等价类。

（3）如果输入条件是一个布尔值，则可以确定一个有效等价类和一个无效等价类。

（4）如果规定了输入数据的一组值，而且程序要对每个输入值分别进行处理，这样可以为每一个输入值确定一个有效等价类，此外，可以针对这组值确定一个无效等价类。

（5）如果规定了输入数据必须遵守的规则，则可以确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

（6）如果确知已划分的等价类中各元素在程序中的处理方式不同，则应将此等价类进一步分成更小的等价类。

3）测试用例的选择

（1）为每一个等价类规定一个唯一的编号。

（2）设计一个新的测试用例，尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止。

（3）设计一个新的测试用例，覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

2. 边界值分析

边界值分析是对等价类划分方法的补充。首先应确定边界情况，通常输入等价类与输出等价类的边界是重点测试的内容，应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。

边界值分析方法选择测试用例的原则在很多方面与等价类划分方法类似。具体原则如下：

（1）如果输入条件规定了值的范围，应取刚刚达到这个范围的边界值以及刚刚超越这个范围的边界值作为测试输入数据。

（2）如果输入条件规定了值的个数，则用最大个数、最小个数、比最大个数多 1、比最小个数少 1 的数作为测试数据。

（3）根据规格说明书的每个输出条件，使用前面的原则（1）。

（4）根据规格说明书的每个输出条件，使用前面的原则（2）。

（5）规格说明书给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

（6）如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界值作为测试用例。

3. 因果图

等价类划分法和边界值分析法都是着重考虑输入条件，但不考虑输入条件之间的各种组合，也不考虑各个输入条件之间的相互制约关系。如果在测试时考虑输入条件之间的各

种组合，则可能的组合数也许是天文数字。因此必须考虑使用一种适合于描述各种条件组合、产生多个相应动作的测试方法，这就需要因果图。

因果图能够帮助测试人员按照一定的步骤，高效率地开发测试用例，以检测程序输入条件的各种组合。它是将自然语言规格说明转化成形式语言规格说明的一种严格方法，可以指出规格说明存在的不完整性和二义性。因果图的基本符号如图 8.5 所示。

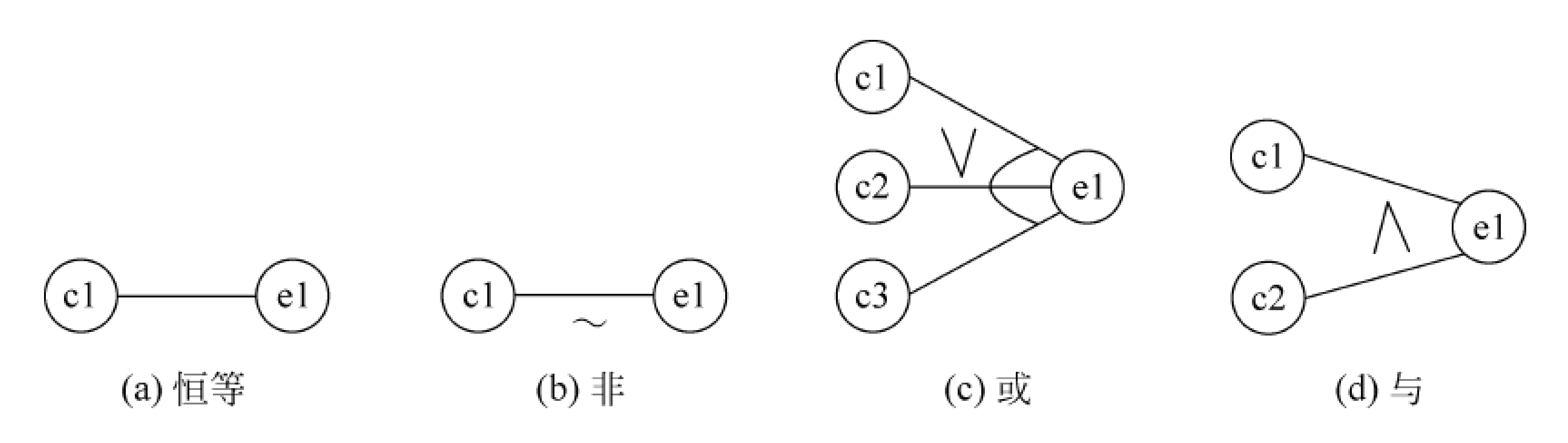


图 8.5 因果图的基本符号

因果图法最终要生成决策表。运用因果图导出测试用例需要以下几个步骤：

(1) 分析程序规格说明中哪些是原因、哪些是结果，原因通常是输入条件或输入条件的等价类，结果是输出条件各个原因与各个结果的"因果图"。

(2) 分析程序规格说明中的语义内容，找出原因与结果之间、原因与原因之间的对应关系，将其表示成连接各个原因与各个结果的"因果图"。

(3) 由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现，为表明这些特定情况，使用一些记号标明约束或限制条件。

(4) 将因果图转换成决策表。

(5) 根据决策表的每一列设计测试用例。

8.3.2 白盒测试用例设计

白盒测试用例设计是从程序内部的逻辑结构出发设计测试用例，因此要求测试用例设计人员对程序的逻辑结构十分清楚，甚至掌握源程序的所有细节。

1. 语句覆盖

语句覆盖(Statement Coverage，SC)是设计若干个测试用例，运行被测试程序，使得每一个可执行语句至少执行一次。这里的"若干个"意味着使用的测试用例越少越好。其公式表示如下：

$$语句覆盖率=(被评价到的语句数量/可执行的语句总数)\times 100\% \qquad (8.1)$$

缺点：对程序执行逻辑的覆盖很低。

2. 判定覆盖

判定覆盖(Decision Coverage，DC)有时也称分支覆盖，就是指设计若干个测试用例，运行被测程序，使得每个判定的取真分支和取假分支至少评价一次。其公式表示如下：

$$判定覆盖率=(被评价到的判定分支个数/判定分支总数)\times 100\% \qquad (8.2)$$

$$判定路径覆盖率(DDP)=(被评价到的判定路径数量/判定路径总数)\times 100\% \qquad (8.3)$$

缺点：主要对整个表达式最终取值进行度量，忽略了表达式内部取值。

3. 条件覆盖

条件覆盖(Condition Coverage，CC)是设计足够多的测试用例，使得每一个判定语句中每个逻辑条件的可能值至少满足一次。其公式表示如下：

$$条件覆盖率=(被评价到的条件取值数量/条件取值总数)\times 100\% \tag{8.4}$$

缺点：不能够满足判定覆盖。

4. 条件判定覆盖

条件判定覆盖(Condition Decision Coverage，CDC)是设计足够多的测试用例，使得判定中每个条件的所有可能(真或假)至少出现一次，并且每个判定本身的判定结果也至少出现一次。其公式表示如下：

$$\begin{aligned}条件判定覆盖率 &= 被评价到的条件取值和判定分支的数量/\\ &(条件取值总数+判定分支总数)\times 100\%\end{aligned} \tag{8.5}$$

缺点：没有考虑单个判定对整体结果的影响，无法发现逻辑错误。

5. 条件组合覆盖

条件组合覆盖也称多条件覆盖(Multiple Condition Coverage，MCC)，是设计足够多的测试用例，使得每个判定中条件的各种可能组合都至少出现一次(以数轴形式划分区域，提取交集，建立最少的测试用例)。其公式表示如下：

$$条件组合覆盖率=(被评价到的条件取值组合数量/条件取值组合总数)\times 100\% \tag{8.6}$$

满足条件覆盖一定满足判定覆盖、条件覆盖、条件判定覆盖。

缺点：判定语句较多时，条件组合值比较多。

6. 路径覆盖

路径覆盖(Path Coverage，PC)是设计足够多的测试用例，执行程序所有可能的路径。其公式表示如下：

$$路径覆盖率=(被执行到的路径数/程序中的总路径数)\times 100\% \tag{8.7}$$

8.4 单元测试

单元是指软件代码的基本组成单位，具有明确的功能、规格定义、与其他部分接口的定义等基本属性，依据这些属性可将程序的不同单元清晰地区分开来。在C语言这样传统的结构化编程语言中，单元一般是函数或子过程；在C++这样的面向对象的语言中，单元一般是类或类的方法。

单元测试(unit testing)一般由程序员自行完成，因而单元测试大多是从程序内部结构出发设计测试用例，采用白盒测试方法。当有多个程序模块时，可以并行独立地开展测试工作。

8.4.1 测试环境

单元测试一般应紧接在编码之后，当源程序编制完成并通过复审和编译检查后，便可开始单元测试。因为单元本身不是一个完整的程序，必须为单元测试模块开发一个驱动模块（driver）和（或）若干个桩模块（stub）。

驱动模块有时也称为“主程序”，调用被测试模块，接收测试数据并将这些数据传递到被测试模块，被测试模块被调用后，“主程序”打印相关结果。桩模块用来模拟测试模块工作过程中所调用的模块，由被测试模块调用，一般只进行很少的数据处理，目的是检验被测试模块与其下级模块的接口。

测试用例设计应与复审工作相结合，根据设计信息选取测试数据将增大发现各类错误的可能性。在设计测试用例的同时，应给出期望结果。

单元测试环境如图 8.6 所示。

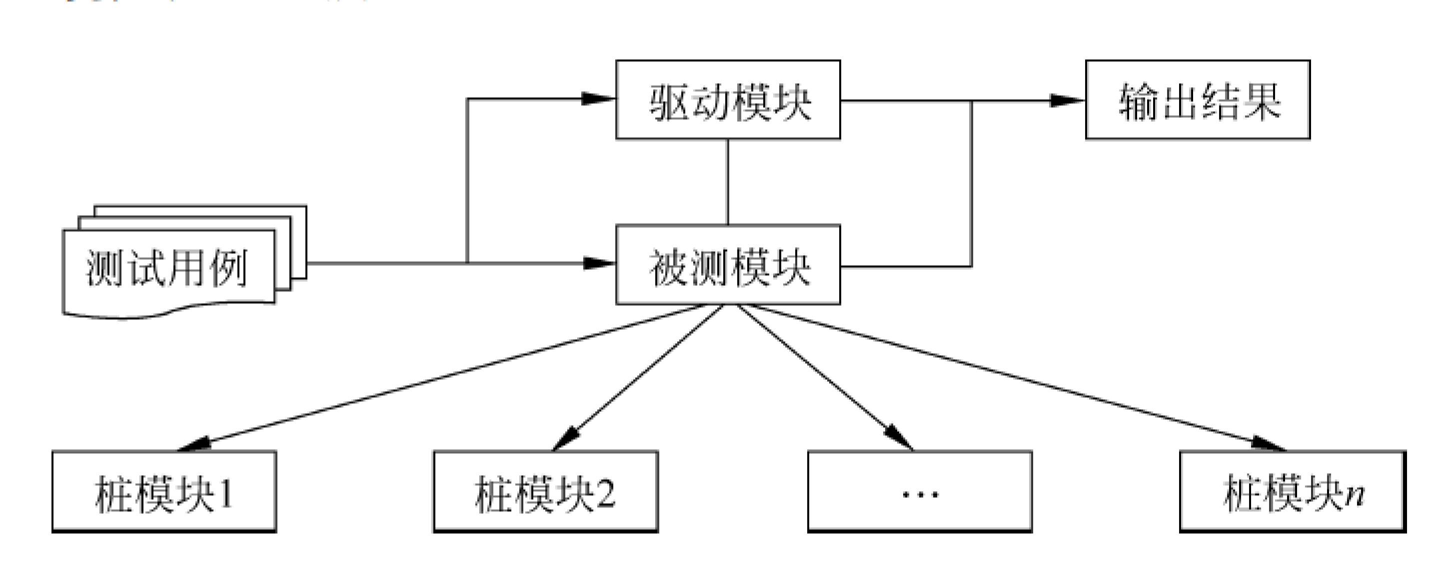


图 8.6 单元测试环境

驱动模块和桩模块是测试使用的模块，而不是软件产品的组成部分，但需要一定的开发费用。若驱动模块和桩模块比较简单，实际开销相对较低。如果仅用简单的驱动模块和桩模块不能完成某些模块的测试任务，这些模块的单元测试可以等到集成测试时进行。

提高模块的内聚度可以简化单元测试，如果每个模块只能完成一个功能，所需测试用例数目将显著减少，模块中的错误也更容易发现。

8.4.2 测试内容

单元测试时，测试者依据详细设计说明书和源程序清单，了解模块的 I/O 条件和逻辑结构。一般可从五个方面进行检查和测试，如图 8.7 所示。

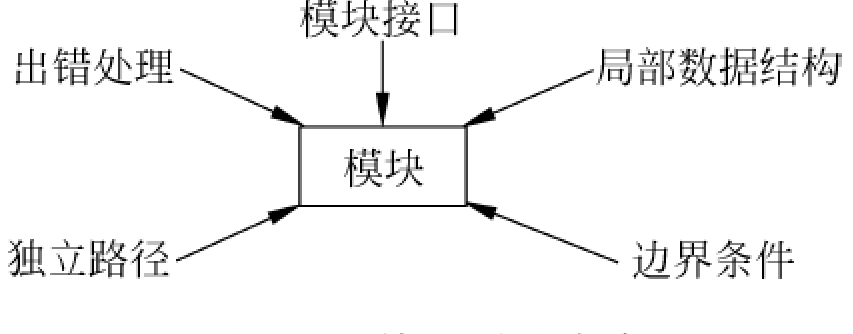


图 8.7 单元测试内容

1. 模块接口

模块接口测试是单元测试的基础。只有在数据能正确流入、流出模块的前提下，其他测试才有意义。测试接口正确与否应该考虑下列因素：

（1）输入的实际参数与形式参数的个数是否相同。

（2）输入的实际参数与形式参数的属性是否匹配。

（3）输入的实际参数与形式参数的量纲是否一致。

(4) 调用其他模块时所给实际参数的个数是否与被调模块的形参个数相同。

(5) 调用其他模块时所给实际参数的属性是否与被调模块的形参属性匹配。

(6) 调用其他模块时所给实际参数的量纲是否与被调模块的形参量纲一致。

(7) 调用预定义函数时所用参数的个数、属性和次序是否正确。

(8) 是否存在与当前入口点无关的参数引用。

(9) 是否修改了只读型参数。

(10) 各模块对全程变量的定义是否一致。

(11) 是否把某些约束作为参数传递。

如果模块内包括外部输入输出,还应该考虑的因素包括文件属性是否正确、Open/Close 语句是否正确、格式说明与输入输出语句是否匹配、缓冲区大小与记录长度是否匹配、文件使用前是否已经打开、是否处理了文件尾、是否处理了输入输出错误、输出信息中是否有文字性错误。

2. 局部数据结构

检查局部数据结构是为了保证临时存储在模块内的数据在程序执行过程中完整、正确。局部数据结构往往是错误的根源,应仔细设计测试用例,力求发现下面几类错误:

(1) 不合适或不相容的类型说明。

(2) 变量无初值。

(3) 变量初始化或默认值有错。

(4) 不正确的变量名(拼写错误或不正确的截断)。

(5) 出现上溢、下溢和地址异常。

3. 边界条件

边界条件测试是单元测试中最重要的一项任务。因为软件经常在边界上失效,采用边界值分析技术,针对边界值及其左、右设计测试用例,很有可能发现新的错误。如果在单元测试中忽略边界条件测试,那么以后测试很难发现问题,即使被发现后对其跟踪,寻找根源,也是很不容易的工作。

4. 独立路径

在模块中应对每一条独立执行路径进行测试,单元测试的基本任务是保证模块中每条语句至少被执行一次。设计测试用例是为了发现因错误计算、不正确的比较、不适当的控制流等造成的错误,此时基本路径测试和循环测试是最常用且最有效的测试技术。

(1) 计算中常见的错误包括误解或用错了运算符优先级、混合类型运算、变量初值错误、精度不够、表达式符号错误。

(2) 比较判断与控制流常常紧密相关,测试用例还应发现的错误包括不同数据类型的对象之间进行比较;错误地使用逻辑运算符或优先级;因计算机表示的局限性,期望理论上相等而实际上不相等的两个量相等;比较运算或变量出错;循环终止条件有可能不出现,陷入死循环;迭代发散时不能退出;错误地修改了循环变量。

5. 出错处理

一个好的设计应能预见各种出错条件，并预设各种出错处理。出错处理同样需要认真测试。测试应着重检查下列问题：

(1) 输出的出错信息难以理解。

(2) 记录的错误与实际遇到的错误不相符。

(3) 在程序自定义的出错处理阶段运行之前系统已介入。

(4) 异常处理不当。

(5) 错误陈述中未能提供足够的定位出错信息。

8.5 集成测试

集成测试(integrated testing)也称组装测试或综合测试。在单元测试的基础上，将所有模块按照设计要求组装成子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独工作，但并不能保证连接起来也能正常工作。程序在某些局部反映不出来的问题在全局上很可能暴露出来，影响功能实现。

8.5.1 测试过程

集成测试过程如图 8.8 所示。

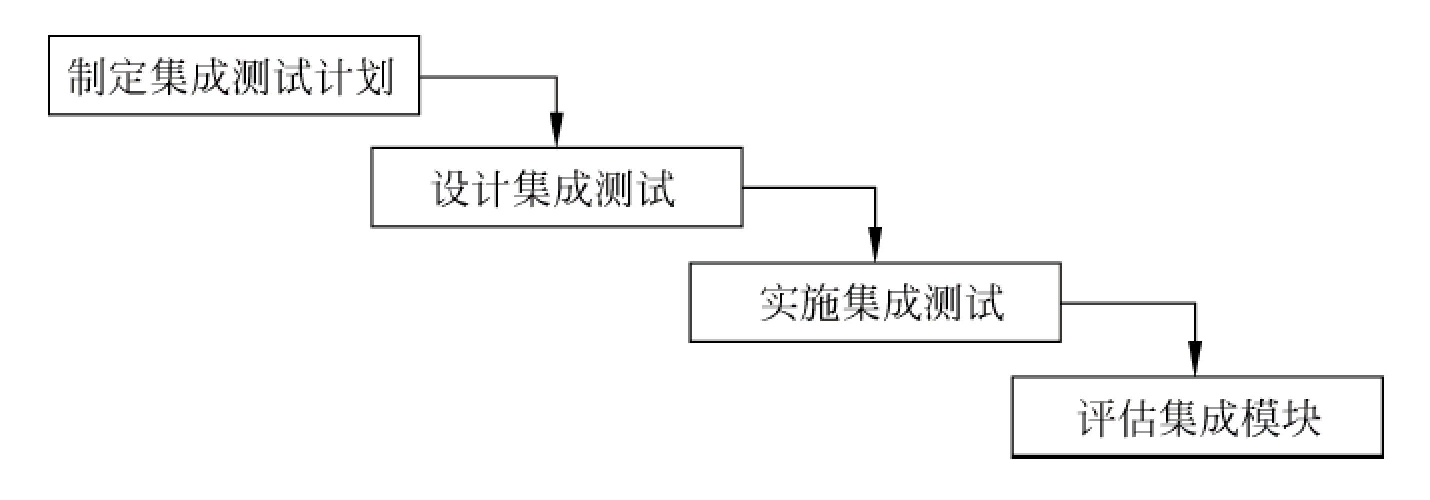


图 8.8 集成测试过程

在执行集成测试过程中，应注意以下问题。

(1) 在制定测试计划时，应考虑以下因素：

① 采用何种集成策略进行集成测试。

② 集成测试过程中连接各个模块的顺序。

③ 模块代码编制和测试进度是否与集成测试的顺序一致。

④ 测试过程中是否需要专门的硬件设备。

解决了上述问题之后，就可以编制测试计划，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期以及需要的测试用例和所期望的测试结果。

(2) 设计和实施集成测试时，应该考虑以下问题：

① 把各个模块连接起来时，穿越模块接口的数据是否会丢失。

② 各个子功能组合起来能否达到预期要求的父功能。

③ 一个模块的功能是否会对另一个模块的功能产生不利影响。

④ 全局数据结构是否有问题。

⑤ 一个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

(3) 判定集成测试是否完成，可按以下几个方面检查：

① 是否成功地执行了测试计划中规定的所有集成测试。

② 是否修正了发现的错误。

③ 测试结果是否通过了专门小组的评审。

集成测试应由专门的测试小组进行，测试小组由有经验的系统设计人员和程序员组成，整个测试活动要在评审人员出席的情况下进行。

在完成预定的集成测试工作之后，测试小组应负责对测试结果进行整理、分析，形成测试报告。测试报告要记录实际的测试结果、在测试中发现的问题、解决问题的方法以及解决之后再次测试的结果。此外，还应提出目前不能解决、需要管理人员和开发人员注意的一些问题，提供测试评审和最终决策，以便提出处理意见。

8.5.2 集成策略

集成测试按单元组装的策略可以分为增量集成测试和非增量集成测试两大类，增量集成又可分为自顶向下集成测试和自底向上集成测试两类。在测试软件系统时，应根据软件特点和工程进度，选用适当的测试策略，有时混合使用增量集成测试的两种策略更为有效，上层模块用自顶向下的方法，下层模块用自底向上的方法。

1. 非增量集成测试

非增量集成测试是将所有模块按层次结构图组装到一起进行测试，最终得到所要求的软件。其目的是尽量缩短测试时间，用最少的测试用例验证系统。

1) 优点

(1) 可以并行测试所有模块，充分利用人力、资源，加快进度。

(2) 需要的测试用例少。

(3) 测试方法简单易行。

2) 缺点

(1) 不能对各模块间的接口进行充分测试，易漏掉潜在接口错误。

(2) 不能很好地对全局数据结构进行测试。

(3) 集成模块过多会出现大量错误，难以定位修改，往往需要经过多次测试才能运行成功。

(4) 软件可靠性难以得到保证。

3) 适用情况

(1) 只需修改或增加少数几个模块的前期产品稳定的项目。

(2) 有少量模块且经过充分测试的小项目。

(3) 基于严格的净室软件工程开发的产品和开发质量较高的产品。

2. 自顶向下集成测试

自顶向下集成测试是构造程序结构的一种增量式方法，从主控模块开始，按照软件的控制层次结构，以深度优先或广度优先的策略，逐步把各个模块集成在一起。深度优先策略首先是把主控制路径上的模块集成在一起，至于选择哪一条路径作为主控制路径，带有随意性，一般根据问题的特性确定。以图 8.9 为例，若选择了最左边一条路径，首先将模块 M_1、M_2、M_5 和 M_8 集成在一起，再将 M_6 集成起来，然后考虑中间和右边的路径。广度优先策略沿控制层次结构水平地向下移动。仍以图 8.9 为例，首先把 M_2、M_3 和 M_4 与主控模块 M_1 集成在一起，再将 M_5、M_6 和其他模块集成起来。

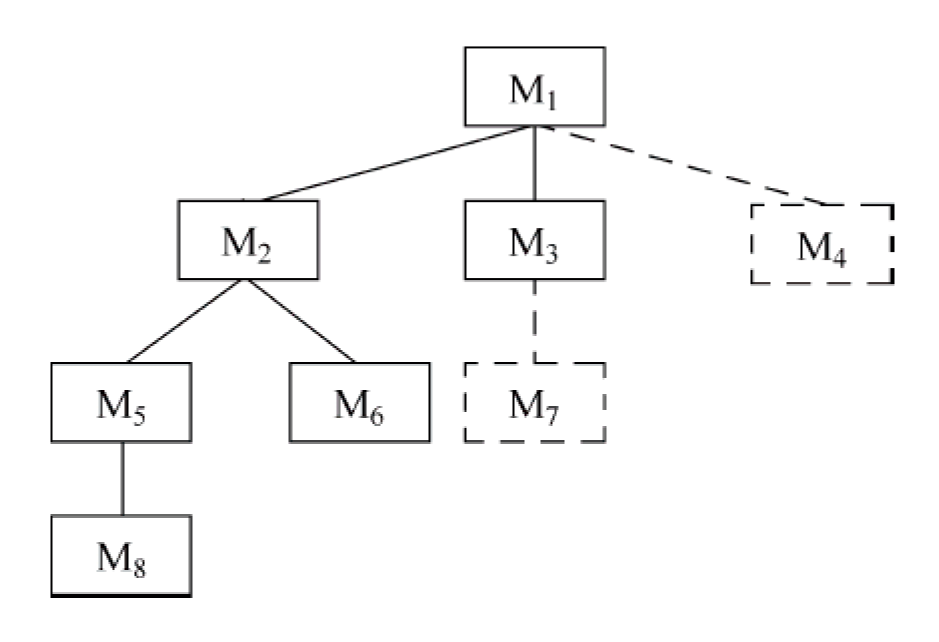


图 8.9 自顶向下集成测试示例

自顶向下集成测试的步骤为：

(1) 以主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代。

(2) 依据所选的集成策略（深度优先或广度优先），每次只替代一个桩模块。

(3) 每集成一个模块立即测试一遍。

(4) 只有每组测试完成后，才着手替换下一个桩模块。

(5) 为避免引入新错误，必须不断地进行回归测试（即全部或部分地重复已做过的测试）。

从第(2)步开始，循环执行上述步骤，直至整个程序构造完毕。

自顶向下集成测试的优点是：能尽早对程序的主要控制和决策机制进行检验，因此较早地发现错误。其缺点是：在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，因此测试并不充分。解决这个问题有几种办法：第一种是把某些测试推迟到用真实模块替代桩模块之后进行；第二种是开发能模拟真实模块的桩模块；第三种是自底向上集成模块。第一种方法又回退为非增量式的集成方法，使错误难以定位和纠正，并且失去了在组装模块时进行一些特定测试的可能性；第二种方法要增加开销；第三种方法比较切实可行，下面专门讨论。

3. 自底向上集成测试

自底向上集成测试是从“原子”模块（即软件结构最低层的模块）开始组装测试，因测试到较高层模块时所需的下层模块功能均已具备，所以不再需要桩模块。

自底向上集成测试的步骤为：

(1) 把低层模块组织成为实现某个子功能的模块群(cluster)。

(2) 开发一个测试驱动模块，控制测试数据输入和测试结果输出。

(3) 对每个模块群进行测试。

(4) 删除测试使用的驱动模块，用较高层模块把模块群组织成为完成更多功能的新模块群。

从第(1)步开始循环执行上述各步骤,直至整个程序构造完毕。图 8.10 说明了上述集成测试过程。

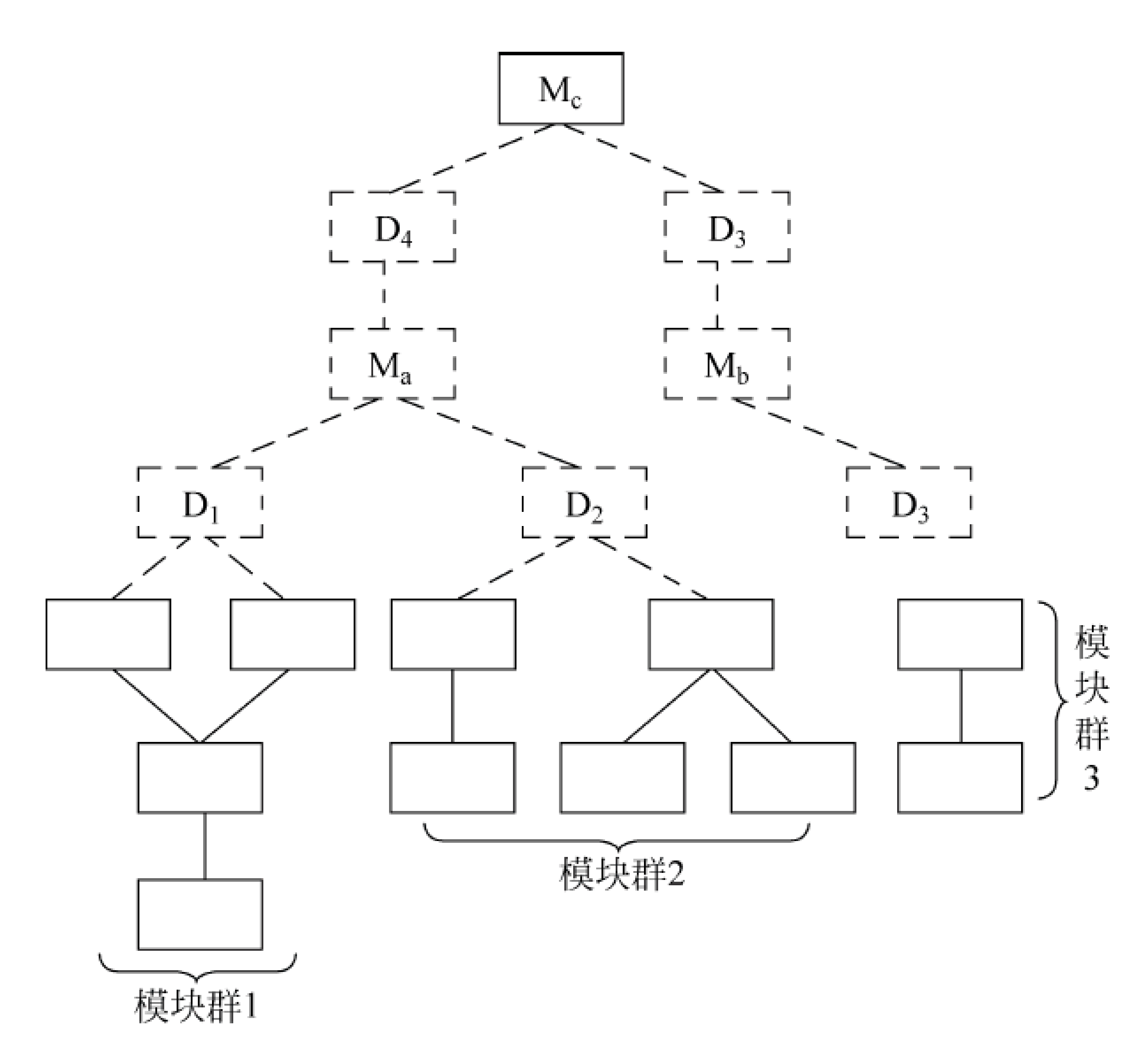


图 8.10　自底向上集成测试示例

首先,“原子”模块被分为三个模块群,每个模块群引入一个驱动模块进行测试。因模块群 1、模块群 2 中的模块均隶属于模块 M_a,因此在驱动模块 D_1、D_2 去掉后,模块群 1 与模块群 2 直接与 M_a 接口;可将 D_3 去掉,M_b 与模块群 3 直接接口,对 M_b 进行集成测试。最后,M_a、M_b 和 M_c 全部集成在一起进行测试。

自底向上集成测试的优点是不使用桩模块,测试用例设计相对简单;缺点是程序最后一个模块加入时才具有整体性。自底向上集成测试与自顶向下集成测试方法优缺点正好相反。

8.6　确认测试

实现软件确认要通过一系列黑盒测试。确认测试(validation testing)同样需要制定测试计划和过程,测试计划应规定测试的种类和测试进度,测试过程则要定义一些特殊的测试用例,旨在说明软件与需求是否一致。无论是计划还是过程,都应该着重考虑软件是否满足合同规定的全部功能和性能,文档资料是否完整、准确,人机界面和其他方面(可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。

8.6.1　测试步骤

确认测试过程如图 8.11 所示。首先要进行基本内容测试以及软件配置审查,然后进行验收测试和安装测试,在通过了专家鉴定之后,才能成为可交付的软件。

基本内容测试在 8.6.2 小节讲述,其他主要工作如下:

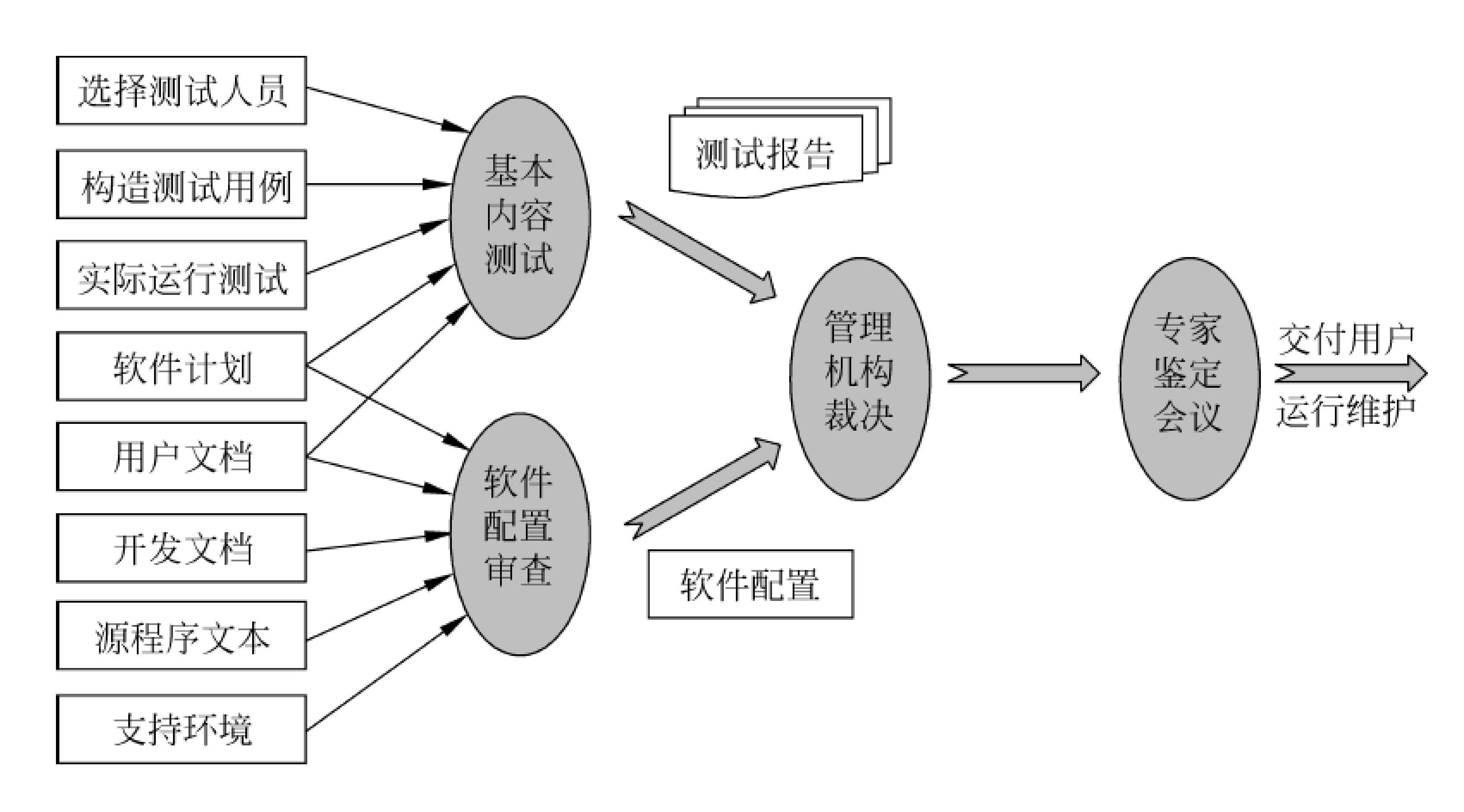


图 8.11 确认测试过程

1）软件配置审查

软件配置审查的目的是保证软件配置的所有成分齐全，各方面的质量符合要求，具有维护阶段所必需的细节，而且已经编排好分类目录。

除了按合同规定的内容和要求，由人工审查软件配置之外，在确认测试过程中应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误，并且适当地补充和改正。

2）验收测试

在通过了系统的基本内容测试及软件配置审查之后，就应开始进行验收测试。验收测试是以用户为主的测试。软件开发人员和 QA（质量保证）人员也应参加验收测试。由用户参加设计测试用例，通过用户界面输入测试数据，并分析测试输出结果。一般使用工作中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误恢复功能等进行确认。

8.6.2 测试内容

在软件开发过程中，确认测试属于测试阶段，在软件集成测试完成且将分散开发的单元构成一个完整的程序之后，才能开始确认测试工作。基本内容测试的一般步骤为：

（1）研究软件需求规格说明，确定软件的功能和性能，明确软件确认测试环境要求。

（2）根据软件的功能和性能，设计测试用例。

（3）根据软件测试环境要求，构建软件确认测试环境。

（4）按照软件确认测试计划，执行确认测试，并记录测试结果。

（5）分析测试结果，编写软件确认测试分析报告。

确认测试的基本内容包括功能测试、性能测试、强度测试和安全性测试等。

1. 功能测试

考察软件对功能需求的完成情况，这是最基本的测试，设计测试用例应该使需求规格的每一项软件功能得到执行和确认。在功能测试中，既要用需求要求的基本数据类型和数据

值进行测试，以测试软件在正常条件下的能力，也要用一系列真实的数据类型和数据值进行测试，以测试软件在超负荷、饱和及其他“最坏情况”下的结果，还要用假想的数据类型和数据值进行测试，以测试软件排斥不规则输入的能力。在测试中，测试用例必须覆盖每个功能的合法边界值和非法边界值。

2. 性能测试

检验软件是否达到需求规格说明中规定的各类性能指标，并满足与性能相关的约束和限制条件。性能测试包括测试软件在获得定量结果时程序计算的精确性；测试在有速度要求时完成功能的时间；测试软件完成功能时处理的数据量；测试软件、硬件中的某些因素是否限制了产品的性能；测试产品的负载潜力；测试程序运行占用的空间。

3. 强度测试

强度测试是在高负荷情况下进行的测试，相当于硬件中的应力测试，要求软件必须被强制在设计能力的极限状态下运行，进而超过此极限，并验证在饱和点的降级不是灾难性的。强度测试在某种程度上可看做是性能测试的延伸，以测试出软件功能、性能的实际极限。强度测试在负载不定的或交互式的、实时的及过程控制测试中很有用处。例如，若雷达控制系统要求跟踪目标区最多可达 200 架飞机，那么强度测试就要测试在目标区已存在 200 架飞机且在第 201 架飞机进入目标区时系统的反应，以及大量飞机在同一时刻进入目标区时的反应。

对性能进行的强度测试包括以下内容：

(1) 提供要求处理的信息量，并超过设计允许的最大值。

(2) 数据传输能力的饱和试验，要求比设计能力传输更多数据。

(3) 存储范围(例如缓冲区、表格区和临时信息区等)超过额定大小时的能力。

(4) 在相当长的时间内软件保持在极限状态下运行的能力。

对降级能力进行的强度测试是指由于计算机部分硬件失效而在设计上允许降级运行的系统，对每种可能的降级方式都必须仔细地加以验证。可以通过将硬件进行实际物理降级的办法实现。例如，切断某设备或部件的电源，以验证失效处理的正确性，包括回到正常运行方式的能力；又如，提高或降低供给某设备或部件的电压，以验证在设备电压不稳定时系统的反应。

在强度测试中，还应测试计算机处于过载状态时系统的反应。有些情况是软件在实际使用中可能遇到的情况，有些情况也许是永远不会发生的情况。但这并不意味着对这些情况的测试没有用途，因为这种不可能遇到的测试情况也会发现软件错误。

4. 安全性测试

安全性测试分为两种：一种称为数据安全性测试(data security testing)；另一种称为软件安全性测试(software security testing)。

(1) 数据安全性是指对硬件、软件进行的保护，以防止受到意外或蓄意的存取、使用、修改、毁坏或泄密。数据安全性也涉及对人员及数据、通信以及计算机安装的物理保护。通过数据安全性测试，验证设置在软件内部的保护和检查机构能否对系统进行防护，使之免受各

种侵害。数据安全性测试就是要设计一些测试用例来破坏软件的保护和检查机构。比如，破坏操作系统的内存保护机构，或破坏数据库管理系统的数据保密机构。

(2) 软件安全性是指软件运行不引起系统事故的能力。软件安全性测试的目的是验证软件在各种规定的条件下不会引起系统事故的能力。

8.7 系统测试

软件只是计算机系统的一个组成部分，需要与系统其他部分(硬件、网络、数据等)组合起来才能真正发挥作用，因此需要进行系统集成和测试。

系统测试(system testing)是将已经确认的软件、计算机硬件、外设、网络、数据和人员等元素结合在一起，进行整个应用系统的测试，目的是在真实的系统工作环境下检验软件是否能同系统其他元素正确地连接，是否满足软件任务书规定的功能和性能要求，发现系统与用户需求不符或矛盾的内容，从而提出更加完善的方案。

8.7.1 特点与方法

1. 系统测试的特点

从软件测试角度来看，系统测试的特点如下：

(1) 系统测试环境是软件真实运行环境最逼真的模拟。在系统测试中，各部分研制完成的真实设备逐渐代替了模拟器或者等效器，是软件从未有过的运行环境。有关真实性的一类错误，包括外部设备接口、输入输出、多处理器设备之间接口不相容，整个系统时序不匹配等，在这种运行环境下能够得到比较全面的暴露。

(2) 通常系统测试的难点是不容易由系统目标直接生成测试用例。这是因为系统任务定义阶段仅能指出软件应该做什么和如何做，并没有给出软件功能的详细描述。软件功能是通过系统任务到软件需求的转换而产生的，过程可能存在错误。系统测试通常由系统人员组织，从系统完成任务的角度进行测试，软件在系统测试环境下针对整个系统要完成的功能进行直接测试，这对检验软件是否满足系统的任务要求非常有意义。

2. 系统测试的方法

系统测试通常以系统人员为主，软件研制人员参加，并就软件需要在系统测试中加强或补充的测试工作提出建议，最后由系统人员确定系统测试内容。系统测试中使用的技术手段一般由系统人员根据总体的系统测试内容确定，软件研制人员可以向系统人员提出软件运行信息、计算结果、性能指标等数据测量和采集要求，并提出一些具体的实施方案，在同系统人员商议之后执行。在系统测试环境下，不可能逐一测试软件功能。软件人员应以情景测试和运行测试为基本方法，尽可能将希望的测试特性形成完整的情景和运行方式，以便和系统整个测试过程较好地融合。为了能够分析运行结果，除了收集完整准确的运行数据，观察、记录执行情况之外，准确记录测试运行环境也非常重要。在系统测试环境中，由于其他设备的加入使得这个因素更加突出。上述情景和运行测试均属于功能测试，使用随机测试也是系统测试的一种方法。

8.7.2 外部接口测试

外部接口测试是系统测试的重点测试项目，也是系统测试要解决的基本问题。外部接口测试主要检验系统各组成部分之间接口的正确性和协调性，也包括检验软件与各系统组成部分之间接口的正确性和协调性。具体包括：测试软件对系统每一个真实接口的正确性；检查从接口接收和发送数据的能力；检查数据格式、数据类型的符合性；检查数据的约定、协议的一致性；检查软件对外围设备接口特点的适应性，如在允许的频率、幅度变化范围内，软件接口部件能否正常工作；在异常情况（例如持续高或低电平、故障状态下、数据混乱、状态混乱等）下，软件接口的反应和处理方式。测试时应设法检验硬件接口在已知失效模式下软件的反应。为了验证软件接口和设备接口物理意义的一致性，要使用一定的测试用例，以便在接口的物理意义不一致时能暴露错误。例如，使接口传送的数据取满量程值，以便在输入后通过结果来判别软件对该接口意义的理解与外围设备接口物理意义的一致性。接口测试应持续一段时间，以观察软件的处理速度或数据采集速度是否与外围设备的数据传送速度相匹配，既不会丢失数据，也不会重复采集数据。

8.7.3 其他测试类型

在系统测试中，除了进行外部接口测试外，还应有选择地完成下列测试类型：

（1）功能测试。对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测试需求。测试目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。基于黑盒技术，通过图形用户界面（GUI）与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。

（2）性能测试。确保系统在正常的工作量和预期的最繁重工作量下能够正常工作。性能评测是一种性能测试，对响应时间、事务处理速率和其他与时间相关的需求进行评测和评估。性能评测的目标是核实性能需求是否都已满足。实施和执行性能评测的目的是将测试对象的性能行为作为条件（例如工作量或硬件配置）的一种函数来进行评测和微调。

（3）业务周期测试。确保测试对象及背景的进程都按照要求的业务模型和时间表正确运行。业务周期测试应模拟在一段时间内项目执行的活动。应先确定一个时间段（例如一年），然后执行将在该时间段发生的事务和活动。这种测试包括所有的日、周和月周期，以及所有与日期相关的事件。

（4）争用测试。测试指定事务在多用户条件下，发生死锁、死锁条件和并行控制等问题。在一台或多台计算机上同时运行 GUI 和虚拟用户，模拟实际用户环境。争用测试的主要技术是通过多个用户同时对相同资源进行操作。争用测试的完成标准是多个事务或多个用户在可接受的时间范围内成功地完成测试，没有发生任何故障。

（5）负载测试。负载测试是一种性能测试。在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关

的问题。

(6) 强度测试。强度测试是一种性能测试。实施和执行此类测试的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足,测试对象就可能表现出一些在正常条件下并不明显的缺陷。而其他缺陷则可能是由于争用共享资源(例如数据库锁或网络带宽)而造成的。强度测试还可用于确定测试对象能够处理的最大工作量。

(7) 容量测试。容量测试是使测试对象处理大量的数据,以确定是否达到了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如,如果测试对象正在为生成一份报表而处理一组数据库记录,那么容量测试就会使用一个大型的测试数据库,检验该软件是否正常运行并生成了正确的报表。

(8) 安全性和访问控制测试。安全性和访问控制测试侧重于安全性的两个关键方面:一是应用程序级别安全性,包括对数据或业务功能的访问;二是系统级别安全性,包括系统登录或远程访问。应用程序级别安全性可确保在预期的安全性情况下,主角只能访问特定的功能或用例,或者只能访问有限的数据;系统级别安全性可确保只有具备系统访问权限的用户才能访问应用程序,而且只能通过相应的网关来访问。

(9) 故障转移和恢复测试。故障转移和恢复测试可确保测试对象能成功完成故障转移,并能从导致意外数据损失或数据完整性破坏的各种硬件、软件或网络故障中恢复。故障转移测试可确保对于必须持续运行的系统,一旦发生故障,备用系统就将不失时机地"顶替"发生故障的系统,以避免丢失任何数据或事务;恢复测试是一种对抗性的测试过程,将把应用程序或系统置于极端的条件下(或者是模拟的极端条件下),以产生故障(例如设备输入输出故障或无效的数据库指针和关键字等),然后调用恢复进程并监测和检查应用程序和系统,核实应用程序、系统和数据是否已得到了正确的恢复。

(10) 配置测试。配置测试核实测试对象在不同的软件和硬件配置中的运行情况。在大多数实际环境中,客户机工作站、网络连接和数据库服务器的具体硬件规格会有所不同。客户机工作站可能安装不同的软件,如应用程序、驱动程序等,而且在任何时候,都可能运行许多不同的软件组合,从而占用不同的资源。

(11) 安装测试。安装测试有两个目的:

一是确保软件在正常情况和异常情况下都能进行安装。例如,进行首次安装、升级、完整的或自定义的安装。异常情况包括磁盘空间不足、缺少目录创建权限等。

二是核实软件在安装后是否可立即正常运行,通常是指运行大量为功能测试制定的测试。

思考题

1. 理解软件测试过程流程。
2. 简要叙述软件测试的原则。
3. 理解静态测试与动态测试的特点。
4. 静态测试与动态测试分别有哪些方法?
5. 理解黑盒测试与白盒测试,并对两种方法进行比较。
6. 黑盒测试用例设计有哪些技术?

7．边界值分析方法设计测试用例应遵循哪些原则？
8．白盒测试用例设计有哪些具体方法？
9．单元测试要求怎样的环境？单元测试的内容是什么？
10．简述集成测试的过程。
11．集成测试有哪些集成策略？各有何特点？
12．确认测试的基本内容包括哪些？
13．系统测试有何特点？
14．系统测试包括哪些测试类型？每种测试类型的测试内容是什么？



第9章 软件维护

软件维护是软件交付使用后，为保证软件安全稳定运行而对其进行修改的过程。通过修改软件缺陷，提高软件性能或其他属性，使软件产品适应新环境，延长软件生命周期。

随着软件的广泛应用，暴露出的问题日益增多，软件维护变得日趋重要。软件维护是一项耗时间、耗精力的工作。由于用户的计算机应用基础不同，对使用软件的熟悉程度不同，软件与实际工作的结合情况不同，造成软件使用过程中出现各种各样的问题。研究表明，软件维护是软件生命周期中花钱最多、耗时最长的阶段，软件机构可能要将60%～80%的精力用在现有软件维护上。

本章讲述软件维护的相关内容，目的是掌握软件维护方法，提高软件可维护性，减少软件维护工作量，降低软件系统总成本。

9.1 软件维护概述

9.1.1 软件维护的内容

软件维护的内容有以下几个方面：

(1) 源程序维护。由于计算机是按源程序的逻辑功能处理正常业务，当发生错误以及数据或业务发生变化时，源程序要适应这些变化，通常需要修改并重新编译来实现，这是软件维护的主要工作。

(2) 数据维护。软件系统运行离不开数据，数据通常以数据文件或数据库文件方式保存，当系统业务处理对数据需求发生变化时，增加、删除、修改数据内容，建立新的数据文件等事项经常发生。

(3) 代码维护。随着软件版本的变更，旧代码将不适应新要求，必须修改旧的代码系统或制定新的代码系统。

(4) 环境维护。计算机技术的发展使得硬件产品更新速度加快，与之相配套的软件系统也要随着变化，如操作系统和数据库升级等。若要保证系统的先进性，延长软件生命周期，必须考虑软硬件环境对系统的影响，通过修改软件或改变环境来适应这些变化。

9.1.2 软件维护的分类

国际电工委员会根据软件维护工作的需要，将软件维护分为如下五类：

(1) 修复性维护。其目的是消除软件中有影响的潜在错误，从而根除失效根源或故障，确保软件完全符合需求规格说明中的功能描述，并充分满足需求规格说明中表示的用户需求。当一个有影响的潜在错误引起的软件故障被检测出来时，就可能启动修复性维护。

(2) 预防性维护。其目的是在潜在错误变成现行错误之前，预防性地检测或更正这些错误。通过预防性维护，提高软件的可移植性，实现编码元素的最优化，改进软件的抗干扰能力。预防性维护通常在固定的时间间隔定期进行。

(3) 完善性维护。其目的是提高软件的质量和效率，具体可以是减少存储空间，减少等待或响应时间，减少运行时间，有效地格式化输入数据或测试结果，增加预处理程序或后台处理程序等。完善性维护的范围比预防性维护的范围小，不需要制定维护计划或进行长时间的可行性分析。进行完善性维护的最终目的是提高软件竞争力或响应用户需求。

(4) 适应性维护。其目的是使软件便于移植到其他机器上，或在相同的机器上安装更先进的系统软件版本。当硬件或软件控制的部件改变时，需要修改软件接口。引起适应性维护的原因通常是为了适应软件供方或硬件供方的变化，或者是为了满足管理部门为潜在市场而制定的策略。

(5) 进化性维护。其目的是对软件进行修改、更新和扩充，以使软件尽可能满足用户要求。进化性维护通常是由于用户不满意或者要求更好的软件性能而引起的。在进行进化性维护可行性分析时，必须使新功能与现有功能融为一体，绝不能反过来影响软件的行为、性能或质量。

在 GB/T14070—1993《软件维护指南》中，对上述五类软件维护活动类型的分类进行了合并，将进化性维护和完善性维护合并为完善性维护，将预防性维护和修复性维护合并为改正性维护。因而将软件维护工作分为如下三类：

(1) 完善性维护。完善性维护是为扩充功能和改善性能而进行的修改和扩充，以满足用户变化了的需求。主要内容包括：为扩充或增强功能而做的修改(如扩充解决范围、算法优化等)；为提高性能而做的修改(如提高精度、节省存储空间等)；为便于维护而做的修改(如增加注释、改进易读性等)。

(2) 适应性维护。适应性维护是为适应软件运行环境变化而做的修改。变化的主要内容包括：影响系统的规定、法律和规则的变化；硬件配置(如机型、终端、打印机等)的变化；数据格式或文卷结构的变化；系统软件(如操作系统、编译系统或实用程序等)的变化。

(3) 改正性维护。改正性维护是为维持系统正常运行，对在开发过程中产生而在测试和验收时没有发现的错误而进行的改正。必须改正的错误包括设计错误、逻辑错误、编码错误、文档错误、数据错误等。

根据国外统计，完善性维护占全部维护工作的 54%～72%，适应性维护占全部维护工作的 18%～25%，改正性维护占全部维护工作的 17%～21%。

9.1.3　软件维护的要求

在 GB/T 8566—2001《信息技术软件生存周期过程》中，有关软件维护的要求有如下 24 点：

(1) 维护者应为实施维护过程的活动和任务制定并执行计划和规程，并形成文档。

(2) 维护者应建立接受、记录和跟踪来自用户的问题报告和修改请求，以及向用户提供

反馈的规程。无论何时遇到问题,都应记录下来并纳入问题解决过程。

(3) 维护者应实施配置管理过程的接口(或建立与配置管理过程的组织接口),以管理对现有系统的修改。

(4) 维护者应分析问题报告或修改请求,以确定在类型(如纠正、改进、预防和对新环境的适应)、范围(如修改规模、涉及的费用、修改时间)和关键性(如对性能、安全或保密性的影响)等方面对组织、现有系统和接口的影响。

(5) 维护者应重现或验证问题。

(6) 维护者应根据分析考虑实施修改的方案。

(7) 维护者应将问题、修改请求、分析结果和实施方案形成文档。

(8) 维护者应按合同的规定使选定的修改方案得到批准。

(9) 维护者应进行分析并确定需要修改的文档、软件单元和版本,并将这些结果形成文档。

(10) 维护者应进入开发过程,以便实施修改。开发过程的需求补充如下:应规定测试和评价系统中已修改部分与未修改部分(软件单元、部件和配置项)的准则,并形成文档;应确保完整并正确地实现了新的和已修改的需求;同时确保原来的、未修改的需求不受影响;测试结果应形成文档。

(11) 维护者应与授权修改的组织一起实施评审,确定已修改系统的完整性。

(12) 维护者应按合同规定,促使完成的修改得到批准。

(13) 如果一个系统或软件产品(包括数据)从一个旧的运行环境移植到一个新的运行环境,应确保在移植期间产生或修改的任何软件及数据遵循国家标准。

(14) 应制定、文档化和实施一个移植计划,策划活动应包括用户。计划应包括移植需求分析和定义、移植工具开发、软件产品和数据变换、移植执行、移植验证,以及未来对旧环境的支持。

(15) 应将移植计划和活动通知用户。通知内容应包括:为何不再支持旧环境的说明;对新环境及其可用日期的描述;一旦对旧环境的支持取消,如果有其他可用的支持方案,应进行描述。

(16) 旧环境和新环境可能并行工作,以便平稳地过渡到新环境。在此期间,应按合同规定提供必要的培训。

(17) 当预定的移植到来时,应通知所有相关方。所有相关旧环境的文档、日志和编码应放入档案中。

(18) 应进行运行后的评审,以评估变更到新环境的影响。评审结果应送到相应的权威机构,以提供信息、进行指导和采取措施。

(19) 根据合同关于数据保护和审核的需求,旧环境使用的数据或与旧环境有关的数据应是可访问的。

(20) 应制定退役计划,以撤销运行和维护组织,并将退役计划形成文档。策划活动应包括用户。计划应予以执行,并包括如下内容:在一定时期之后终止全部或部分支持,归档软件产品及其相关文档;负责未来的后续支持事项;适当时转换为新的软件产品;可访问数据的归档副本。

(21) 用户应得到退役计划和活动通知。通知应包括如下内容:替代或升级及其可用日期的说明;为何不再支持该软件产品的说明;在撤销支持时,其他可用支持方案的说明。

(22) 退役软件和新软件应并行工作,以便平稳地过渡到新系统。在此期间,应按合同规定为用户提供培训。

(23) 当预定的退役到来时,应通知所有相关方。适当时,所有相关的开发文档、日志和编码都应放入档案。

(24) 根据合同关于数据保护和审核的需求,退役软件产品使用的或与退役软件产品有关的数据应是可访问的。

9.2　软件维护过程模型

软件维护过程模型是对软件进化过程的抽象表示,有助于分析软件维护期间的活动。采用何种维护模型,应了解各种模型的特点,并根据维护环境来决定。以下分析常见的几种模型。

9.2.1　快速修改模型

快速修改模型表示的维护过程是一种"救火"方法,即软件维护过程中临时定制的方法,软件出现问题后尽可能快地解决,不对长期效应进行分析就实施修改。通常不分析修改的波及效应对代码结构产生的影响,即使进行分析也极少写入文档。快速修改模型结构如图 9.1 所示。

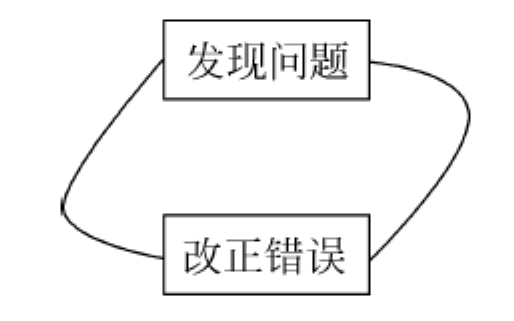


图 9.1　快速修改模型

在适当的环境中,这种模型非常有效。例如,如果系统是由一个人开发和维护,这个人对系统非常了解,有能力在没有详细文档的情况下管理系统,能就是否修改和如何修改做出判断,维护工作快速、经济。

在有很多客户的商业运行环境中,这种方式并不可靠,但仍有很多机构使用这种模型,是由于软件维护受到时间和资源的限制。例如,客户要求改正一个错误,但不愿意等待软件公司烦琐的更改过程和风险分析。

如果软件长时间地依靠快速修改,就会累积很多问题,软件就会越来越难以维护,维护成本就会增加,就会丧失在最初阶段使用快速修改模型得到的任何优势。为解决这个问题,采用的策略是把快速修改模型集成到另一个更精细的模型中,快速修改作为一种在外界压力下应急的更改方式,修改完成后,再根据精细模型要求采取一定的措施。

9.2.2　Boehm 模型

Barry W. Boehm 博士根据经济学模型和原理,提出了维护过程模型,即 Boehm 模型。Boehm 模型的理论是经济学模型和原理不仅可以改进维护的生产率,而且还有助于理解维护过程。Boehm 模型如图 9.2 所示。Boehm 模型将维护过程分为管理层决策、实现更改、软件交付和评估四个阶段,维护过程表示为闭合环路,通过管理层决策推动维护过程。在管理层决策阶段,运用具体策略并对所提出的一组更改进行费效评估,确定一组经过批

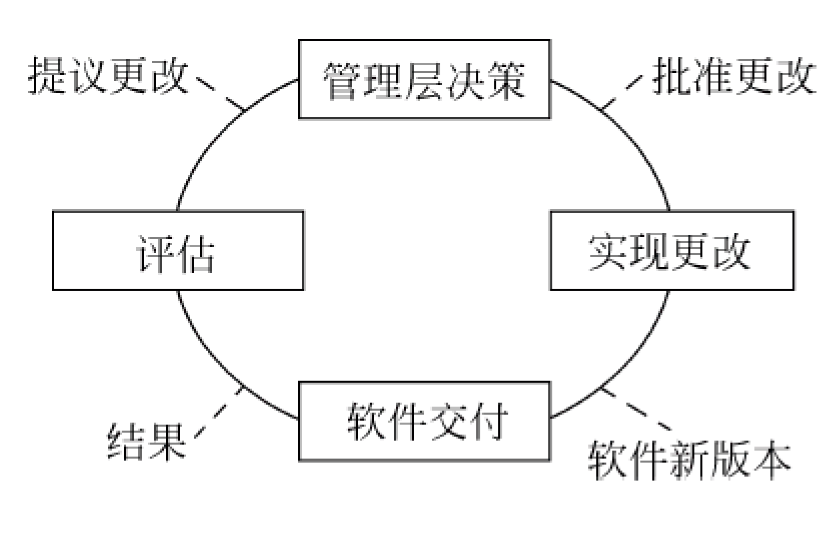


图 9.2　Boehm 模型

准的更改以及实施更改的专用预算。

从生产功能上看，该模型反映了投资与收益之间的经济学关系，反映出一种典型的阶段图：

(1) 投资阶段。这是低资源投入和低收益阶段，对应于紧急修改和增强有强烈要求的新发布软件产品。

(2) 高回报阶段。机构通过软件产品得到不断增长的回报，初始问题被解决。在这个阶段中，资源投入到文档与效率的提高上，机构的积累效益迅速增长。

(3) 效益减少阶段。到了一定的时间点，积累效益增长速率逐渐变慢。产品处于效用的顶峰时，到达更改变得越来越不经济的阶段。

Boehm 模型侧重于管理层决策，根据批准的更改实施更改，使维护活动在投资和效益之间取得平衡，从经济利益角度来驱动软件维护过程。基于这个过程，可以使组织制定合理的维护策略，做出满足组织效益的维护决策。

9.2.3 IEEE 模型

随着软件行业的发展，人们逐渐认识到对软件进行标准化维护的重要性。因此，IEEE 计算机学会软件工程标准化分会颁布了“IEEE 软件维护标准”(IEEE 12191993)，详细阐述了管理与执行软件维护活动的迭代过程，包括软件维护的输入、处理、控制及输出等。该标准指出，应该在规划软件开发的时候就进行软件维护规划。IEEE 模型如图 9.3 所示。

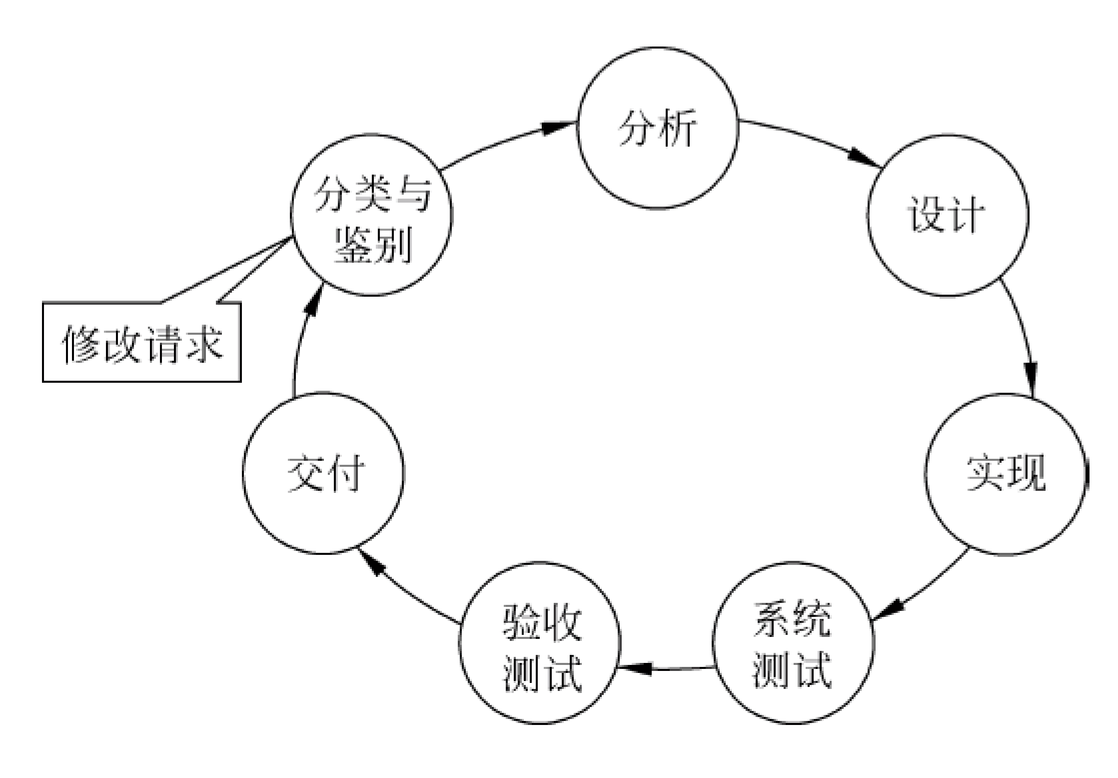


图 9.3 IEEE 模型

IEEE 模型的各阶段详细说明如下：

(1) 分类与鉴别。软件维护起始于由用户、开发人员或管理人员提出的对软件的修改请求，并以修改请求单的形式提交。修改请求可以是任何维护类别(改正性维护、适应性维护、完善性维护、预防性维护)，由维护机构确定是何种类型，划分到相应的维护类别中，并确定处理的优先级别。每一个申请都分配一个唯一编号，请求内容输入到数据库系统中，以便进行跟踪。收集、审查度量和指标要求从这个阶段开始。

(2) 分析。首先进行维护的可行性分析，然后进行详细分析。可行性分析主要确定更改对软件的影响、可行的解决方案以及所需的费用等内容；详细分析主要是提出完整的更改需求说明，鉴别需要更改的要素(模块)，提出测试方案或策略，制定实施计划。最后由配置控制委员会(CCB)审查并决定是否着手开始工作。

(3) 设计。汇总全部信息以便审查并用于软件更改设计，包括项目文档、分析阶段结

果、源代码、知识库等信息。分析阶段应更新设计基线，更新测试计划，修订详细分析结果，核实维护需求。

（4）实现。制定更改计划以便进行更改。主要包括如下过程：编码与单元测试、集成、风险分析、测试审查准备、更新文档等。

（5）系统测试。主要测试程序之间的接口，以确保系统满足原来的需求以及新增加的更改需求。同时进行回归测试，确保不引入新错误。

（6）验收测试。在完成系统集成后，由用户或第三方完成验收测试。主要完成如下工作：报告测试结果，进行功能配置审核（确定系统功能满足需求），建立软件新版本（或基线），准备软件文档最终版本（包括系统文档和用户文档）等。

（7）交付。修改后的系统交给用户安装并运行。应进行物理配置审核、文档备份、安装与培训等工作。交付后，系统开始投入使用。

与前面的模型相比，IEEE 模型详细规定了软件维护过程的各种活动，作为标准，可以适用于所有的软件维护过程。但是对于不同的软件，因特性不同，维护过程也会有所区别，IEEE 模型是一个大而全的规范，针对不同的软件，应该对过程进行剪裁，同时细化一些过程的任务。

9.2.4　迭代增强模型

最初该模型是作为一种开发模型提出的，因为软件开发人员通常不能充分理解需求，不能构建完善系统，因此很适合维护。提出该模型的基础是在软件生命周期内对软件实施的更改，是一种迭代过程，并以迭代方式增强软件系统。

模型要求有完备的文档作为每轮迭代的开始，实际上是三个阶段的循环，如图 9.4 所示。

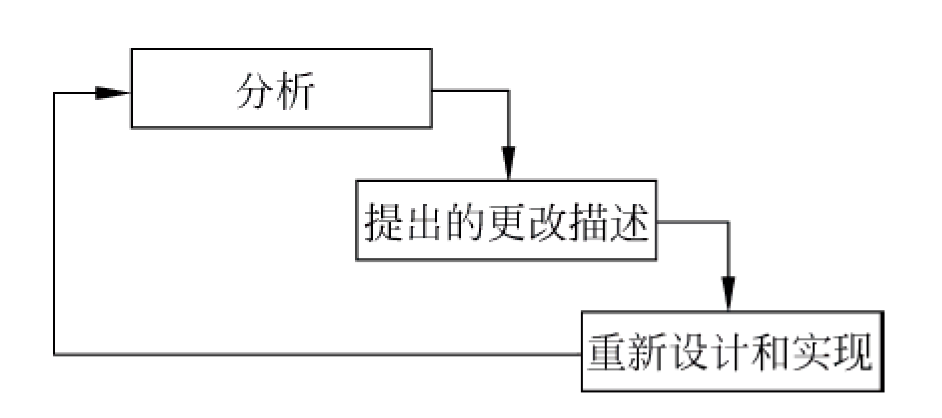


图 9.4　迭代增强模型

根据更改请求对文档的影响，首先修改每个阶段（需求、设计、编码、测试和分析）的文档，这种修改在整套文档中传播，并重新设计系统。

根据维护环境找出快速解决方案。通常使用“最快”的解决方案会引出很多问题，迭代模型本身也同化其他模型，因此可以在结构化的环境中集成快速修改模型，进行快速修改，找出问题，并在下一轮迭代中专门解决这些问题。

9.2.5　维护模型分析

（1）快速修改模型只适用于“救火”的情形，不能保证软件维护过程高可靠性的要求。

（2）Boehm 模型适用于维护组织的管理层，是从经济利益角度驱动软件维护过程，侧重于规定批准更改的策略，对于更改实现过程规定不多。Boehm 模型同样不能保证软件维护过程高可靠性的要求，而且对于软件维护而言，重点关注维护质量和可靠性，而不是批准更改的策略。

（3）IEEE 模型是一个大而全的规范，适用于所有的软件维护过程。通过系统测试和验收测试保证维护质量。

（4）迭代增强模型对软件更改需求并不充分，通过迭代不断地完善软件。

除了以上介绍的几种模型外，还有 Osborne 模型、面向重用模型等。不同模型的侧重点不同，有的关注经济学问题，有的关注产品，有的关注过程。所有模型都有优缺点，没有一种模型适用于所有情况，而且往往将模型组合在一起是最好的解决方案。确定维护模型以后，为了实现有效的维护过程管理，还应该深入理解软件过程及相关概念。

9.3 软件维护技术

进行软件维护需要一系列相关的技术支持，简要说明如下：

(1) 重新设计工程(re-engineering)。使用重新构造设计技术再次产生新的源代码的过程叫做重新设计工程。重新设计工程不仅可以恢复系统的详细设计，而且可以从已有的系统中借鉴可用的组件。

(2) 逆向工程(reverse engineering)。当前的维护活动都是关注代码级别的维护。但是如果相关的文档不完整、不准确或已过时，对于代码的理解是比较困难的。为了解决这些问题，逆向工程利用程序转换过程从现有的程序代码中获取程序概要说明，主要关注应用程序中的重要属性。

(3) 程序转换(restructuring)。软件系统不断升级，软件就会越来越复杂，理解和维护也会日益困难，采用程序转换的方法可以使软件从一种表现形式转换为另外一种形式，并且不改变其功能和语义。程序转换和重新设计工程有一定的联系，重新设计工程是程序在形式上的转换，而程序转换是代码级别的转换。

(4) 应用程序理解(application understanding)。在软件维护过程中应用程序理解是最耗时的一个阶段，因为需要阅读文档、阅读源程序、理解源程序等。逆向工程和应用程序理解最为接近，逆向工程是通过提供系统的说明模型来帮助理解程序，而应用程序理解采用自底向上、自顶向下、动态分割、静态分割等技术来帮助理解程序。

(5) 影响分析(impact analysis)。影响分析的目标是分析采用新措施时对软件维护的影响程度，估计新措施带来的成本和风险。

(6) 回归测试(regression testing)。回归测试是指当对某项功能进行修改后，其他未修改组件或模块的功能不会受到影响、不会引入新的错误、不会产生未预料的负面效果。回归测试主要集中在现有系统的可靠性和耗费影响分析维护上。

(7) 组件技术(component based software engineering)。软件重用是减少软件成本的有效技术。当采用重用组件时，需要建立或购买重用库，然后进行分类、注册和检索。重用的过程必须集成到软件开发生命周期和维护模型中，做到和其他组件无缝结合，共同完成软件功能。

(8) 软件配置管理(SCM)。软件配置管理的目标是管理软件开发和维护过程，控制软件变化，加强软件管理的可视性和对软件的跟踪，确保软件开发质量。SCM 是软件开发和维护的基础。一个 SCM 是一个系统的缩略图，包括采用的技术、用户需求、角色、数据库、过程模型、用户培训和管理决策等。

(9) 基于 WWW 的维护(WWW based maintenance)。基于 WWW 的软件维护是指通过 Internet 获得大量有用信息来支持软件维护，软件维护者可以在 WWW 基础上实现软件维护和开发。通过驻留在客户端的 Agent 收集软件错误信息，并加以分析处理，形成软件故障报告，通过 Internet/LAN 发送到软件维护中心，使得软件维护者可以迅速掌握软件最

新维护资料，确保维护的高效率。

(10) 维护过程模型(maintenance process model)。为了更好地控制软件维护，需要把软件维护过程分为几个独立的阶段。目前已有许多研究机构提出了不同的软件维护模型，帮助维护人员进行维护和管理。

(11) 维护标准(maintenance standard)。1993 年 IEEE 第一次发布了关于软件维护的 IEEE 标准(IEEE 1219)，为软件维护者提供了软件维护的通用框架和过程。1995 年 ISO/IEC 也发布了 ISO/IEC 12207 关于软件维护的过程和标准。

9.4　软件维护过程

软件维护工作按先后顺序可分为准备、提出需求、需求分析、分析评审、修改实现、测试、验证、升级等过程。

1. 准备

充分准备是维护工作的良好开端。准备包括以下工作：

(1) 指定维护人员。维护人员不仅要有专业技术素质，还要有与人沟通、与客户打交道的服务素质。

(2) 建立通畅方便的维护通信渠道。通信渠道包括网站、电话、电子邮件等方式，有条件的还要开通免费服务电话。

(3) 培训。培训包括技术培训和与客户沟通技巧等方面的培训。

(4) 编制和批准《软件维护计划》。计划要明确人员和职责、通信渠道、维护费用估算等。

2. 提出请求

软件维护起始于一个对系统的更改请求，通常请求由用户、现场维护工程师或开发人员以问题报告单的形式提出。对于用户提出更改请求的情况，一般是用户直接向技术支持人员或现场维护工程师口头提出，需要技术支持人员或现场维护工程师与用户沟通，了解用户的实际需求，生成问题报告单。问题报告单的参考格式如表 9.1 所示。问题不仅仅包括故障，同时也包括新增需求和功能增加。对于每一个申请都必须分配一个唯一的编号。

表 9.1　问题报告单

提交人员		提交日期	年　月　日		
审核人员		审核日期	年　月　日		
初始编号		问题编号			
问题内容					
系统环境					
近期操作					
初步诊断					
目前处理方法					
目前系统状态					
严重程度		解决时限			
提出人员		确认人员		分析人员	

提出请求的主要活动包括以下几个方面：

(1) 填写问题报告单。问题报告单由用户或者现场维护工程师根据出现的问题填写，或者开发人员根据软件产品内部改进的需要填写。

(2) 向管理者提交问题报告单。

(3) 管理者确认问题报告单，并确定过程活动中的角色和任务分工。

(4) 指定的维护人员开始跟踪维护过程，生成系统跟踪文档，通过在不同阶段更新状态来跟踪此次维护。

3. 需求分析

需求分析是指主要负责人即分析人员，对系统更改请求即问题报告单进行分析，包括问题定位，问题涉及的具体产品及相应的修改规模，给出具体的解决方案、问题涉及的文档，提出测试方案及策略等，最终形成问题分析报告。分析人员对系统要有全局性认识，要理解主要功能模块之间交互的整体情况。问题分析报告的参考格式如表9.2所示。

表9.2 问题分析报告

提交人员		提交日期	年 月 日		
审核人员		审核日期	年 月 日		
初始编号		分析人员			
涉及产品					
分析内容					
分析结果					
解决方案					
测试方案					
预计测试时间					
预计解决时间					
需要修改文档					
解决人员		建议评审人员		评审时间	
评审意见		评审人员签字		管理者签字	

需求分析的主要活动包括以下几个方面：

(1) 分析人员根据问题报告单和相关产品手册讨论分析问题，在分析结束后提交相应的问题分析报告。分析的内容主要如下：

① 类型。判断解决此问题的维护类型是纠正、改进、预防还是对新环境的适应。

② 范围。确定问题涉及的具体产品及相应的修改规模。

③ 解决。判断此问题是否与以前的某个问题完全相同，即根据问题现象判断问题是否是已经发现的问题。如果尚未发现过，需要生成一个新的问题编号，确定解决人员，预计解决时间，提出解决方案，若暂时不能提出彻底的解决方案，应提出目前的应对方案。如果此问题已出现过，应在问题分析报告中进行标注，给出曾经出现问题的编号和相关文档。

④ 文档。判断源程序修改涉及哪些文档修改。

⑤ 测试。如果需要测试，确定测试的内容和目的、测试方案以及预计测试时间。

(2) 此流程结束后，维护人员在系统跟踪表中更新状态。

4. 分析评审

分析评审是保证维护质量的重要手段，可以使存在的问题尽早发现，降低维护后期发现问题的风险。分析人员根据问题的难易程度、问题修改涉及面的大小以及对解决问题的把握能力等因素来决定采取哪种评审方式。

评审方式分为两种：

(1) 正式评审。正式评审需要严格的评审过程，对于问题影响较大、问题较难、分析人员对问题把握不够的情况，需要按照正式评审方式进行。

(2) 非正式评审。非正式评审不需要严格的评审流程，分析人员直接发送检查包，等待评审人员审查回复即可。对于问题影响不大、问题比较容易解决的情况，可以采用这种评审方式。

正式评审流程如图 9.5 所示，各步骤工作如下：

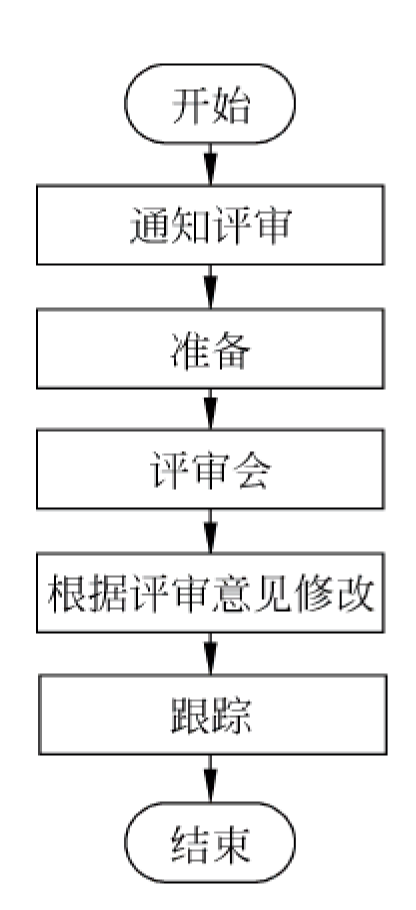


图 9.5 正式评审流程

(1) 通知评审。管理者和分析人员一起选择评审人员，由管理者指定评审负责人；分析人员将待评审的软件产品作为检查包打包，发送给评审负责人；评审负责人确定评审会时间、地点、设备，根据问题情况确定评审准备时间，从而确定评审会时间；评审负责人发布评审通知。

(2) 准备。评审人员认真审阅待评审的软件产品，记录审阅过程发现的问题和疑问，并形成文档。评审人员需提前将评审问题记录发给评审负责人。

(3) 评审会。评审负责人主持会议，把握会议进程；评审人员提出疑问和问题；分析人员对疑问和问题做出解释；记录员做详细的问题记录；评审完毕后，评审负责人带领评审人员对评审意见达成一致，做出评审结论。如果通过，不需要复审，但可能需要修改或纠正一些小问题；如果没有通过，需要复审、做重大修改或重新分析。评审组长或记录员会后完成评审记录和评审问题列表，交由所有评审员签字确认。

(4) 根据评审意见修改。评审负责人在评审会结束后将评审问题列表分发给分析人员，并和分析人员确认解决期限；分析人员对照评审问题列表对软件产品进行修改，并在解决期限内完成。

(5) 跟踪。评审负责人在修改期限内对照评审问题列表，检查每个问题是否修改完成。对于分析人员认为不需要修改的问题，判断分析人员的决策是否合理，并修改问题状态(关闭/未关闭)；对于需要复审的工作产品，继续开评审会复审。

分析评审的主要活动包括以下几个方面：

(1) 分析人员确定采用何种评审方式进行评审。

(2) 评审人员对问题分析报告中提出的分析结果的准确性以及解决方案和测试方案的可行性进行评审，具体评审过程依据不同的评审方式确定。

(3) 若通过评审，维护人员在系统跟踪表中更新状态；若没有通过评审，则应对此问题重新进行分析定位。

5. 修改实现

（1）理解程序过程。解决人员根据问题报告单、问题分析报告和相关产品手册对产品进行修改。修改软件时，首先要理解程序，目的是帮助完成要求的更改。理解程序时，必须注意以下几点：

① 理解程序的功能和目标。

② 掌握程序的结构信息，即从程序中细分出若干结构成分。如系统结构、控制结构、数据结构、输入输出结构等。

③ 了解数据流信息，即数据来源以及在哪里被使用。

④ 了解控制流信息，即执行每条路径的结果。

⑤ 理解程序的操作要求。

一般来说，理解程序包括三种活动：阅读有关程序文档、阅读源代码、运行程序。理解程序过程如图 9.6 所示。

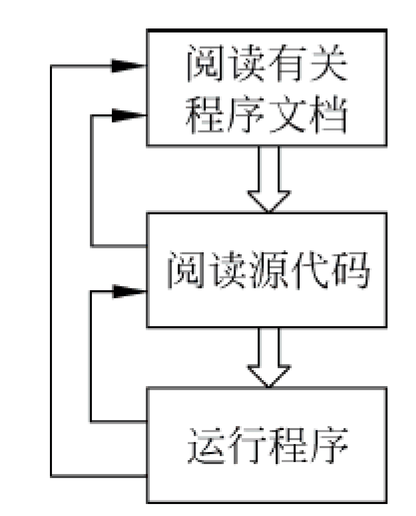


图 9.6 理解程序过程

① 阅读有关程序文档。解决人员要浏览、仔细阅读不同来源的信息，包括系统文档、规格说明和设计文档，建立对系统的总体理解。可以使用附有结构图、数据流图和控制流图的文档。如果系统文档不正确、过时或者不存在，可以省略这个阶段。

② 阅读源代码。完成对程序的全局和局部了解。全局了解可以从顶层理解系统，确定更改可能对系统其他部分带来的影响；局部了解要集中精力到特定的系统部件上，获得有关系统结构、数据类型和算法模式。用于检查源代码的静态分析器等工具可在这个阶段使用，可以产生交叉引用表，指示不同的标识符在程序哪些部分使用。

③ 运行程序。运行程序并跟踪数据，其目的是研究实际程序的动态行为。运行程序的好处是发现通过阅读源代码很难发现的系统的某些特性。

在实践中，理解程序的过程并不一定是以这种有序的方式展开，有时要通过各种活动的迭代和回溯，澄清疑问和获得更多信息。

（2）修改的副作用。根据理解来修改软件，解决人员要了解修改程序时可能带来的副作用，以便在程序修改时注意。副作用包括以下几个方面：

① 修改代码的副作用。修改源代码时，可能引入错误。例如，删除或修改一个子程序，删除或修改一个标号，改变程序代码的时序关系，改变占用存储的大小，改变逻辑运算符，修改文件的打开和关闭，提高程序执行效率，以及把设计上的改变翻译成代码的改变，为边界条件的逻辑测试做出改变，都容易引入错误。

② 修改数据的副作用。在修改数据结构时，有可能造成软件设计与数据结构不匹配，因而导致软件出错。

③ 修改文档的副作用。对数据流、软件结构、模块逻辑或任何其他有关特性进行修改时，必须对相关技术文档进行修改，否则会导致文档与程序功能不匹配、缺省条件改变、新错误信息不正确等，使得软件文档不能反映软件当前状态。

（3）修改实现的主要活动。

① 解决人员理解程序。

② 解决人员提交软件修改申请，申请软件出库。具体过程根据配置管理过程中变更控制流程执行。

③ 如果软件修改申请得到批准，解决人员根据问题报告单、问题分析报告和相关产品手册进行修改，并按照分析定位过程中的文档要求完成相关文档修改，在修改过程中注意不要引入副作用。

④ 修改完成后，解决人员对修改后的代码进行单元测试。

⑤ 单元测试通过后，形成软件修改报告(参考格式如表 9.3 所示)，记录软件修改的情况，同时根据修改后的结果生成软件升级报告，为升级做好准备。

⑥ 维护人员在系统跟踪表中更新状态。

表 9.3 软件修改报告

编 号		提交人		提交日期	
软件名称				软件负责人	
修改开始时间	计划		实际		
计划提交时间			实际提交时间		
修改原因					
修改内容					
改动清单					
版本修改记录	序号	名称	原始版本	当前版本	修改人员
修改结果					
审核人员			审核时间		

6. 测试

软件修改后，测试人员依据问题分析报告中的测试方案进行测试。软件修改容易引入新错误，因此要通过回归测试来减少修改带来的副作用。通过构建一组回归测试用例，覆盖程序中曾经发现的每个错误的测试，以后当这些代码被修改后，都可以运行这些测试用例。在人力资源充足的情况下，建议由独立测试人员对软件修改进行测试；如果人力资源紧张，可以由维护人员进行交叉测试，避免解决人员自己测试自己修改的软件。

测试阶段的主要活动包括以下几个方面：

(1) 测试人员根据问题分析报告中的测试方案对解决人员提交的文档进行审核、对软件进行测试，包括系统测试和回归测试。

(2) 测试完成后，生成测试报告。

(3) 如果测试通过评审，解决人员生成问题解决报告(参考格式如表 9.4 所示)，同时完成升级报告，写明如何进行系统升级，维护人员在系统跟踪表中更新状态；如果测试没有通过评审，则应根据测试问题重新对软件进行修改。

表 9.4 问题解决报告

提交人员		提交日期	年 月 日
审核人员		审核日期	年 月 日
问题编号		解决人员	
实际解决时间		解决结果	
遗留问题			
相关文档			
受影响文档			
通知相关人员			
评审意见			
评审签字		评审时间	年 月 日

7. 验证

软件修改测试结束后提交系统升级之前，要经过验证阶段，即评审人员对问题是否正确解决进行评审。验证的发起者为管理者，目的是保证软件维护质量，可以参照分析评审阶段的方式进行评审。

验证阶段的主要活动包括以下几个方面：

(1) 管理者确定验证活动的方式。

(2) 评审人员对问题解决结果和测试结果进行评审，对象是修改的产品和各种文档。

(3) 评审结束后，评审人员在《问题解决报告》中签署评审意见。如果评审确认问题已经得到解决，维护人员在系统跟踪列表中更新状态；如果未通过评审，则由解决人员对此问题重新修改。

8. 升级

软件在公司内部验证完成后，需要发布给用户，对系统升级。升级过程由开发人员、工程人员及用户共同完成。升级流程图如图 9.7 所示。

(1) 升级提出。由现场维护人员书写系统更改记录，作为一次升级申请。

(2) 升级审批。由维护管理员和管理者审核系统更改记录中的各项内容，以及相关信息是否完备，综合考虑各方面因素，对此次升级进行评估。对于符合升级条件的申请，统一安排升级时间，管理者指定升级负责人，发起升级。推迟或者取消暂时不适合升级的项目。

(3) 升级准备。升级负责人向软件修改部门发出升级请求；修改部门进行代码和文档出库，并执行配置审计，发送给升级负责人；升级负责人进行简单的测试和检查，发现错误后，反馈给相关修改部门重新修改，完成后重新出库；升级负责人再次核查升级内容，直到没有发现错误，确定升级版本。

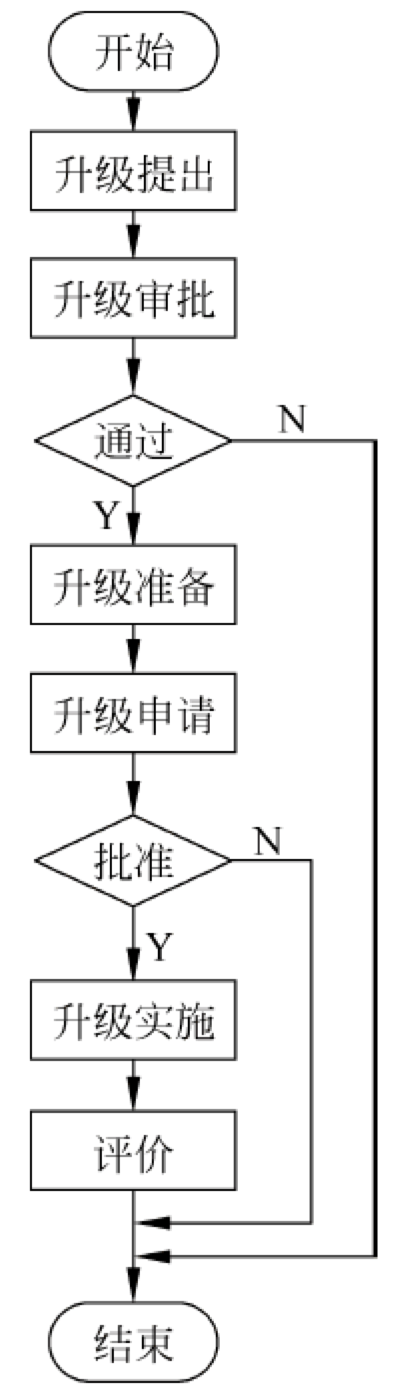


图 9.7 升级流程图

(4) 升级申请。由升级负责人向软件用户提出申请,如果用户批准,负责人则要通知用户做相应准备,以及通知软件升级技术人员做好准备;如果用户不批准,则要弄清原因,以便再次提出申请。

(5) 升级实施。可以按照如下步骤进行:

① 详细阅读升级报告。参加升级工作的人员要对升级报告中的所有描述有明确理解,对升级中使用的指令和参数确认无疑问,对照工程现场的实际情况检查是否满足升级条件。如果对升级报告有疑问应尽早提出,对于不熟悉的命令或参数,应在测试环境下进行测试或者找技术人员确认,由于升级报告延误无法提前确认的更应谨慎操作。

② 检查。检查升级所需要的软硬件是否齐全;如需要用户配合,应该提前通知用户做好相关准备;由升级负责人确定升级执行人员、升级测试人员和升级后的职守人员,使相关人员职责清晰。

③ 做好恢复准备。升级存在风险,升级前要做好数据和系统备份,做好恢复到升级前状态的准备。

④ 执行升级。升级过程严格按照升级报告的要求逐项进行操作,不得遗漏和调换执行次序;如果出现与升级报告不一致的情况,必须与远程技术支持人员沟通,确认后方可进行。

⑤ 监督机制。在人员允许的情况下,最好一人操作一人在旁边监督,并进行最后核查,多人分工操作时要求交叉检查。

⑥ 升级后的测试与例检。测试正确是检验升级后是否正常的直接标准,测试时应该按照测试报告的内容逐项进行,并填写测试报告;升级完成后进行详细的系统例检。这两项工作完成后,向维护管理员提交系统例检表、系统测试表和系统更改记录,由维护管理员对这几份报告进行审核,记录升级情况。

⑦ 升级后的监控。升级负责人安排专人负责升级后的监控工作,对于重大和高风险的升级,要严密监控。

⑧ 远程技术支持。升级责任人、升级执行人和远程技术支持人员进行充分沟通,及时报告同步升级的进展情况。

(6) 评价。升级完成后,要对结果进行评价,便于今后改进。对于升级结果,可分为成功、修改后成功、失败三种状态。当升级失败时,要分析是现场维护的原因、软件修改的原因,还是用户原因。评价还包括系统运行一段时间后的评价。

9.5 软件维护控制

软件维护必须有控制地进行,使整个过程处于适当的管理和控制之下。除了控制预算、进度和人员之外,关键是由软件维护主管负责控制和修改系统。

大量编码在开发过程中并非都考虑到了维护。即使原来是良好设计和实现的编码逻辑,也会因无休止的“快速排错”和修补工作受到破坏。所以,一个系统不仅在开发时要考虑维护,在维护时也要考虑将来的维护。

软件维护的目标是保持系统功能并及时而又满意地响应用户需求。软件系统的可维护性常常随着时间的推移而降低。如果没有为软件维护管理制定严格的条例,或者条例贯彻

不力，许多系统都将蜕变到无法继续维护的地步。

软件维护控制是保持一个有秩序的维护过程，在这个过程中，所有的维护请求要正式提出、评审，给予优先级并安排进度。对软件维护进行控制的具体步骤如下：

(1) 确定软件维护策略。

(2) 评审和评价所有修改请求。

(3) 为软件维护安排进度。

(4) 将代码修改限制在批准的范围内。

(5) 强制实施文档标准和编码规定。

9.6 软件维护组织管理

管理是改进软件维护过程的主要因素之一。管理者必须指导如何维护软件，行使对整个过程的控制，并保证使用高效的软件维护技术和工具。

为确保软件维护成功，维护过程中要有效使用良好的管理技术和方法，必须建立软件维护组织机构。

9.6.1 组织模式

对软件维护做出正确的管理决策并恰当地分配资源，需要理解软件维护过程和组织，分析软件维护过程中采取的组织模式。采用不同的组织模式，维护过程的具体操作也有很大不同，因此应该选择具体的组织模式。目前，在实际操作中有两大类组织模式：一是将开发与维护合并；二是成立独立的维护部门。

1. 将开发与维护合并

根据变更类型、程序模块、活动领域、应用领域和生命周期阶段等因素，将开发与维护合并可分为下列四种模式：

(1) 模块责任制。指派每个成员负责一个模块，模块责任人负责完成需要在该模块中实现的更改。优点是模块责任人对该模块有很高的专门知识；缺点是没有人负责整个软件系统，工作量难以均匀分摊，很难实现增强维护，很难强制执行编码标准。

(2) 更改责任制。不管要修改哪个模块，每个人都负责一项或多项更改。也就是说，责任人要负责更改的分析、规格说明、设计、实现和测试。优点是整个软件系统更有可能符合一套标准，能够保证更改的完整性，更改可以独立地编码和测试，有可能认真进行代码审查；缺点是与模块责任制相比，需要系统的整体知识，培训新员工所需的时间长，个人没有长期责任，但是有一系列临时性责任。

(3) 工作类型。关键特征是根据工作类型，例如分析、设计等进行划分。不同部门作为一个团队工作，但是有明确定义的责任和角色。优点是每个部门成员都能累积细化的知识、培养细化的技能；缺点是不同部门之间协调需要成本。

(4) 应用类型。根据应用领域划分，例如管理信息系统或办公自动化系统。优点是团队成员之间可积累细化的应用知识；与工作类型模式一样，缺点是存在不同应用领域之间

的协调成本。

2. 成立独立的维护部门

成立与开发部门独立的维护部门专门进行软件维护，通常适用于需要维护大量系统，并且随着软件不断运行，维护工作量不断增长的情况。

(1) 优点：

① 工作内容清晰，且有可审计性。

② 开发人员能够集中精力开发新的软件系统。

③ 有利于促进开发完成后的验收测试。

④ 便于提供高质量和最终用户服务。

(2) 缺点：

① 由于地位不同，会有士气低落的危险。

② 当系统安装以后，开发人员可能会遗忘该系统的知识。

③ 开发和维护需要协作时，成本很高。

④ 可能存在冗余的沟通渠道。

在具体的软件组织中，采用哪种组织模式取决于机构规模和维护工作类型等因素。通常情况下，有经济实力的大型机构能够支持两个独立部门；除开发工作外，有很多软件需要维护，最好设立单独的维护部门。但是在具体实践中，很多软件机构都把开发和维护活动结合在一起，甚至没有明确的组织模式，实行“抓着谁就是谁”的工作方式，增加了软件过程管理的难度，维护人员之间相互推诿，维护过程混乱，降低了维护效率。考虑到软件工程的发展状况、软件规模日益增大和社会分工细化、软件维护日益得到重视等因素，成立独立的维护部门是必然趋势。

9.6.2 人员管理

软件维护机构的人员角色由管理者、维护人员、提出人员、分析人员、解决人员、测试人员、评审人员、升级人员等构成，主要任务是审批维护请求、制定并实施维护策略、控制和管理维护过程、负责软件维护审查、组织评审和验收等，确保完成软件维护任务。不同的角色职责不同，维护机构根据实际情况，为具体人员赋予过程角色。各角色及主要职责如表 9.5 所示。

表 9.5 软件维护机构人员角色及主要职责

角 色	主 要 职 责
管理者	• 确认提出人员提交的问题报告单； • 根据问题性质指定相应的分析人员、解决人员和维护人员； • 审核并决定是否批准解决人员提交的软件修改申请； • 协调相关人员工作，包括维护人员、提出人员、分析人员、解决人员和评审人员； • 对实施产品维护流程进行监督
维护人员	• 全程动态跟踪产品维护流程实例的整个过程； • 维护相关文档

续表

角　色	主要职责
提出人员	• 确定系统更改需求的初始编号； • 填写问题报告单； • 提供与问题相关的信息； • 接收来自分析人员和解决人员的反馈，根据需要提供进一步的详细信息
分析人员	• 分析问题并给出分析结果； • 提出解决方案和测试方案； • 填写问题分析报告
解决人员	• 根据问题分析报告中提出的解决方案解决问题； • 向管理者提交软件修改申请； • 填写问题解决报告、软件修改报告、软件升级报告
测试人员	• 对修改后的软件产品进行测试； • 生成测试报告等相关文档
评审人员	• 对问题分析报告中提出的分析结果的准确性进行评审； • 对问题分析报告中提出的解决方案和测试方案的可行性进行评审； • 对问题解决的结果和测试结果进行验证
升级人员	• 向用户提出升级申请； • 负责具体的系统升级操作

软件维护人员的素质对于进行有效维护十分重要，因此应为维护项目选择合格的各级人员。挑选维护人员和进行维护管理的要点如下：

(1) 维护与开发同等重要，同样具有难度。

(2) 维护人员应是合格的、有责任心的人。

(3) 维护不能当做初级人员“放任自流”式的培训。

(4) 全体人员应当轮流分配去做维护和开发工作。

(5) 出色的维护工作应同出色的开发工作一样受到奖励。

(6) 必须强调对维护人员进行良好的培训。

(7) 轮换分配，不应让一个系统或一个系统的主要部分成为某个人的专属领地。

9.7 软件再工程

随着计算机硬件的迅速发展，人们对软件系统也提出了越来越高的要求，从而直接导致软件系统功能复杂、规模庞大，同时加快了软件系统的更新换代时间。早期开发的软件产品已经不能适应现在的需求，从而提出“软件再工程”的概念。正如“软件危机”促进了人们对“软件工程”的重视一样，软件的不断更新也促进了“软件再工程”的研究与发展。

9.7.1 认识软件再工程

软件再工程是为了以新形式重构已存在的软件系统而进行的检测、分析、更替，以及对新形式的实现。这个过程包括逆向工程、文档重构、结构重建、相关转换以及正向工程等。

其目的是理解已存在的软件系统，然后重新设计实现，增强功能、提高性能或降低实现难度，客观上达到维持软件的现有功能并为今后加入新功能做好准备。软件再工程是对成品软件系统进行再次开发，软件维护期的适应性维护、完善性维护和预防性维护都属于再工程范畴。与从无到有的软件开发不同，再工程面对的不是原始需求，而是已经存在的软件系统，是从已经存在的软件起步开发出新软件的过程。

软件再工程通常分为以下三个过程：

(1) 重构(reconstruction)。在不改变原有系统功能的前提下，重新安排程序的逻辑流程和业务流程，使程序复杂性降低或可读性提高。但是无法从根本上改变软件功能，只是对流程进行优化，不涉及系统数据部分，重构后程序可读性更好。

(2) 逆向工程(reverse engineering)。对已有软件系统进行分析，从源代码出发，逐层抽象，明确系统的组成部分及其相互间的关系，最后得到与具体实现无关的高层抽象结果。逆向工程不只是进行维护，也要研究原系统的算法和设计思想。目前主要方法是反编译、反汇编以及对源程序的分析理解。

(3) 正向工程(forward engineering)。从前期工作生成的与具体实现无关的抽象描述开始，一步一步地细化，直至生成可替换旧软件系统的新系统及相关的详细文档。

再工程(reengineering)是对已有软件系统进行分析，并将其重构为新形式代码的开发过程。即：再工程＝逆向工程＋重构＋正向工程。

软件再工程通过对原系统用新的设计思想重新实现，对原有文档进行更新，对原系统功能进行追加或增强，同时通过再工程和再设计，模块划分更合理，接口定义更清晰，文档更齐全，从而更易维护，也提高了系统可靠性。

9.7.2 软件再工程技术

根据用户对现有软件改进要求不同，再工程活动一般可分为系统级、数据级和源程序级三个层次。再工程方法和技术很多，在实际再工程中，可以从不同角度运用再造、再构、再结构化、文档重构、设计恢复、程序理解等再工程方法和技术。

(1) 再造(rebuilding)。以提高可维护性为目的，对系统整体性进行重新构建。可以通过三种方式实现再造：一是完全废弃旧系统；二是保留现存系统；三是二者结合。

(2) 再构(refactoring)。不改变现存软件外部功能，仅修改内部结构，使整个软件功能更强、性能更好。

(3) 再结构化(restructuring)。在同一抽象级上对软件表现形式进行变换。例如从原来的C/S模式转换到B/S模式。

(4) 文档重构(redocumentation)。由源代码生成更加易于理解的新文档。

(5) 设计恢复(design recovery)。恢复设计判断以及得到该判断的逻辑依据。

(6) 程序理解(comprehension)。从源代码出发，研究如何取得程序的相关知识。

重用是软件再工程的灵魂，再工程可以在不同层次重用原软件系统资源，重用那些完善而具有一致性的文档、可读性很高的可维护性程序。软件再工程的发展离不开软件重用技术的应用和发展。

思考题

1. 软件维护的内容包括哪些方面?
2. 软件维护如何分类?各维护类别的具体维护内容是什么?
3. 有关软件维护的要求很多,谈谈你对软件维护要求的理解。
4. 软件维护过程模型有哪些?如何理解各维护模型?对各维护模型进行综合分析。
5. 简要说明软件维护相关的技术支持。
6. 软件维护按先后顺序可分为哪些过程?
7. 如何理解正式评审和非正式评审?
8. 简述正式评审流程以及各工作步骤。
9. 理解程序时必须注意哪些问题?理解程序包括哪些活动?
10. 修改的副作用包括哪些方面?
11. 简述升级流程。升级实施包括哪些具体步骤?
12. 简述软件维护控制的作用及其具体步骤。
13. 如何确定软件维护的组织模式?
14. 挑选维护人员和进行维护管理的要点有哪些?
15. 如何理解软件再工程及其过程?
16. 软件再工程有哪些方法和技术?

管理篇

软件项目管理是软件项目获得成功的重要因素。通过科学管理可以提高资源使用效率，优化资源组合配置，扩大资源利用范围。软件项目管理涉及的内容很多，管理篇讲述与软件工程过程密切相关的六个方面，分别是进度计划管理、质量管理、成本管理、配置管理、文档管理、人力资源管理。



第10章 进度计划管理

凡事预则立，不预则废。这里的"预"就是指计划或策划，计划的重要性对软件企业是不言而喻的。据有关资料统计，在不成功的软件项目中，有50%以上是由于没有计划或计划不周密造成的。对软件开发项目制定科学周密、切实可行的计划，并严格按照计划进行管理，才能真正发挥计划的作用。软件开发计划有很多种，本章讲述软件开发进度计划。

10.1 软件项目进度计划概述

进度是对执行的活动和里程碑制定的工作计划日期表，它决定是否达到预期目的，是跟踪和沟通项目进展状态的依据，也是跟踪变更对项目影响的依据。项目计划就像一张地图，告诉开发人员如何从一个地方到另一个地方，是项目的起始点，是进一步开发的指南。

10.1.1 进度计划的作用

软件项目计划管理在软件开发过程中之所以处于十分重要的地位，是因为软件项目计划体现了对客户需求的理解。为软件工程管理和运作提供可行的计划，是有条不紊地开展软件项目活动的基础，也是跟踪、监督、评审计划执行情况的依据。没有完善的工作计划常常导致事倍功半，也会使项目在质量、工期和成本上达不到要求，甚至造成软件工程失败。因此，制定周密、简洁和精确的软件项目计划是成功开发软件产品的关键。软件项目进度计划有以下几个作用：

(1) 进度保障。使软件项目开发建立在可靠的基础上，并将计划文档化，由开发人员遵循，保证开发进度顺利进行。

(2) 向客户履行承诺。确定软件项目开发的活动和承诺，使软件开发工作有序而协调地开展，以便根据软件计划的资源、约束和能力逐步向客户履行承诺。

(3) 个人工作指南。明确与软件项目相关的组织和个人承诺，将任务责任落实到项目组和个人，从组织管理上保证项目开发成功。

(4) 跟踪检查计划的依据。根据进度计划，对开发过程及进度进行检查和监控，及时发现问题，保证开发顺利进行。

(5) 管理依据。软件开发企业高层领导管理部门经理，部门经理管理项目经理，项目经理管理组长，组长管理组内成员，管理依据之一就是进度计划。

10.1.2 进度计划管理的过程

软件项目进度计划管理的过程可分为六个阶段，如图 10.1 所示。

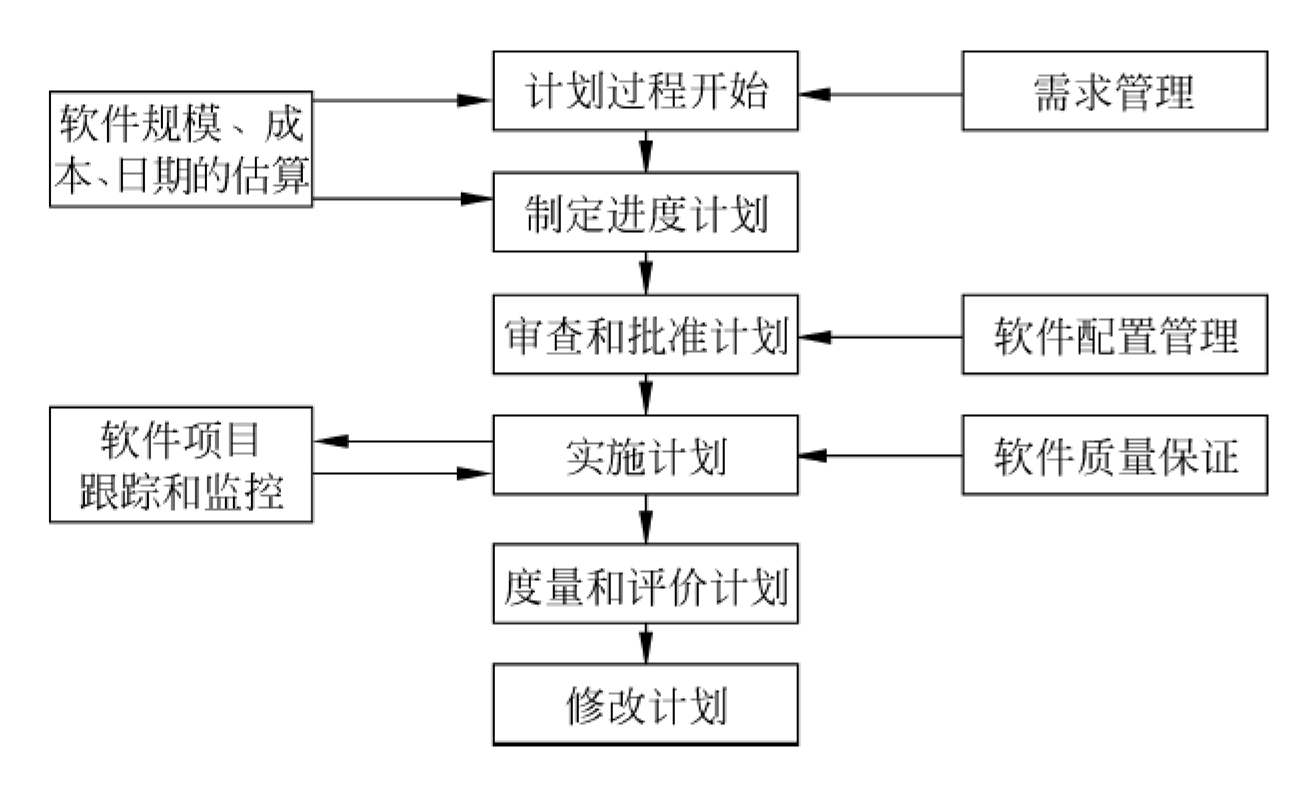


图 10.1 软件项目进度计划管理的过程

(1) 计划初始阶段。根据软件项目开发的内容，结合公司的实际情况，确定项目经理，由项目经理对项目的可行性研究报告、招标投标文件、合同等初期材料进行分析，明确初始需求，对规模、成本、时间、质量等需求进行初步估计，指明项目的初始风险和限制，收集初始计划数据，成立计划组并且指定计划负责人，计划负责人通常由项目经理担任。

(2) 制定软件开发进度计划。如果公司做过类似项目，计划组就应查找以前项目的开发计划案例，选择制定计划的样板，分析案例存在的问题，提出修改意见，把计划样板修改成适合本项目的开发进度计划，提交进度计划草稿。如果公司未做过类似项目，则要选择进度计划的编制方法，根据编制规程进行编制，提交软件进度计划草稿。

(3) 对软件进度计划草稿进行审查和批准。项目计划负责人对计划草稿进行技术检查，对发现的问题提出解决办法，并对草稿作进一步修改和更新，确定无误后，提交给部门经理，由部门经理审查后再进行修改。如果是大型软件项目，进度计划还要提交给总经理和技术总监审查，根据审查意见进行修改。审查通过后，由公司主管批准实施。

(4) 实施软件进度计划。项目相关人员执行软件开发计划规定的任务，开展相应的工作。此过程中要执行软件质量保证，检查软件质量报告。同时进行项目的跟踪和监控，确保计划完成。为确保计划完成，项目经理应对进度计划进行跟踪和监控，随时掌握进度计划的执行情况，解决执行过程中存在的问题。

(5) 计划执行过程的度量和评价。在实施过程中项目相关人员必须严格执行计划，如果确实因计划不合理而不能执行，要及时向项目经理提出意见。项目经理根据开发人员提出的意见，找出进度计划和执行情况的差距，找出造成差距的原因，对计划提出修改意见，估计改进后的效果，为重新修改计划提供根据。

(6) 修改软件进度计划。根据进度计划修改意见，并分析过程改进后的影响，决定是否需要对计划进行修改，提交进度计划问题报告和修改意见。主管人员同意修改后，项目经理根据修改意见进行修改，并提交给相应的主管人员，确定可行后，由公司主管批准实施。

从软件项目进度计划过程中可知，软件项目进度计划的成熟程度是在计划制定与执行

过程中,通过不断总结经验逐步提高的,使软件项目开发过程逐渐趋于成熟。

10.1.3　进度计划管理注意事项

在软件项目进度计划管理中,要注意处理好以下问题:

(1) 全局计划和局部计划的关系。在一个大型软件项目的开发过程中,为了降低开发的复杂性,通常将整个项目分解为若干个子项目,这样就会出现全局计划和局部计划、软件开发进度计划与各种支持保证计划之间的关系问题。处理各种计划之间关系的原则是局部服从全局,协调一致,相互配合,形成一个有机的整体。

(2) 计划的稳定性与灵活性。计划要具有稳定性,不允许在开发过程中轻易改变计划;但计划也要有适当的灵活性,在时间安排上要留有缓冲期。一方面要把开发计划变成组织和个人自觉的行为指南,充分认识执行计划的严格性;另一方面要启发程序员在计划管理的指导下,自由、主动地进行设计和开发,主动完善产品的功能特性,引入新技术、新概念,提高软件产品的质量和开发效益。

(3) 标准化与创新的关系。软件开发进度难以控制,质量也难以保证,软件系统结构复杂,各部分联系密切,这就决定了软件开发很像科学研究过程,既要注意标准化,严格遵循软件工程标准,又要发挥创新精神。但是这种创新必须是在遵守标准的前提下,并且在计划指导下进行的创新,否则这种创新会分散开发人员的精力。

(4) 分工与合作的关系。一个大的软件项目只有进行分工与合作才能顺利完成。分工要做到任务明确、责任清楚,合作就不允许各行其是,要使许多小的、平行的小组或是单个程序员,在项目计划管理下一起协同工作,成为步调一致的整体,形成组织力量。在计划中要对分工与合作进行明确的规定和说明,在执行过程中要加强管理,及时协调,确保同步进行。

(5) 计划的可检查性与评估。软件开发过程中很多活动交互作用,必须精确而形式化地描述,针对关键问题制定详细的实施计划,计划尽可能量化、准确,使计划具有可检查性,以便于跟踪和监控。一个新的项目完成后,要对计划进行评估来分析开发过程中的问题,提出改进意见,进一步完善计划,如此反复,逐步提高软件企业的能力成熟程度。

10.2　进度计划编制方法

10.2.1　甘特图法

甘特图(Gantt chart)也称为条状图(bar chart)。通过条状图来显示项目进度,以及其他与时间相关的、系统进展的内在关系随着时间进展的情况。其中,横轴表示时间,纵轴表示活动(项目或作业)。线条表示在整个期间上的计划和实际的活动完成情况。甘特图可以直观地表明任务计划在什么时候进行,以及实际进展与计划要求的对比。管理者由此可以非常顺利地弄清每一项任务还剩下哪些工作要做,并可评估工作是提前、滞后,还是正常进行。除此以外,甘特图还有简单、醒目和便于编制等特点。所以,甘特图对于项目管理是一种理想的控制工具。

1. 甘特图绘制步骤

(1) 明确项目牵涉的各项活动、任务。内容包括项目名称(包括顺序)、开始时间、工期、任务类型(依赖/决定性)和依赖于哪一项任务。

(2) 创建甘特图草图。这一阶段由于资源局限,可能无法开展项目所有活动。开展某些活动还需要以另外一些活动的顺利完成为前提条件。

(3) 确定项目活动依赖关系及时序进度。使用草图,按照依赖关系类型将活动联系起来,进行安排。此步骤的作用和需要注意的问题如下:

① 保证在未来计划有所调整的情况下,各项活动仍然能够按照正确的时序进行。

② 确保所有依赖性活动能并且只能在决定性活动完成之后按计划展开。

③ 避免关键性路径过长。关键性路径由贯穿项目始终的关键性任务所决定,既表示了项目的最长耗时,也表示了完成项目的最短可能时间。要特别注意,关键性路径会由于单项活动进度的提前或延期而发生变化。

④ 不要滥用项目资源,同时,对于进度表上的不可预知事件要安排适当的富裕时间(slack time)。但是富裕时间不适用于关键性任务,因为作为关键性路径的一部分,它们的时序进度对整个项目至关重要。

(4) 计算单项活动任务的工时量。

(5) 确定活动任务的执行人员及适时按需调整工时。

(6) 计算整个项目时间。专业性软件可以自动完成该项工作。

2. 甘特图绘制举例

某软件开发项目由若干任务组成,说明如下:

(1) 根据软件工程的原理和方法,按照需求分析、软件设计、编码、调试以及试运行的各个步骤安排进度计划,各个步骤的工作通常是交叉进行的。

(2) 调研。项目设计前到相关单位进行详细调研。

(3) 需求分析。产生需求分析报告后要由甲方签字确认。

(4) 设计、代码编写、试运行等阶段时间较长,因为系统涉及多个子系统,要逐个实施。第一个子系统实施成功大约在项目开发后的两个月。

(5) 培训与试运行同时进行。

甘特图的绘制非常灵活。可以是表格形式,也可以是二维坐标形式;可以用条形图表示任务的起止时间,也可以用三角形框表示特定时间;时间表示可以在图的上面,也可以在图的下面;如果开工日期已经确定,时间表示应为具体时间,如果未确定,时间表示应为时间长度,如月、周、日等;活动一般从上到下排列在图的左边。需要确定项目包括哪些活动、活动的顺序以及每项活动持续的时间。下面例子中时间以月为单位,分两种形式绘制甘特图。

形式 1:表格形式、条形框表示起止时间

甘特图作为一种控制工具,帮助管理者发现实际进度偏离计划的情况。用表格形式、条形框表示起止时间的甘特图如图 10.2 所示。图中的有关元素说明如下。

(1) 实心条框:表示计划情况,在项目计划时所有实心条框都已在图中绘制完成。起点表示计划开始时间,终点表示计划结束时间。

单位：月

任务	1	2	3	4	5	6	7	8
调研					今天			
需求分析								
设计								
代码编写								
试运行								
验收								

注释：表示计划情况　表示实际完成情况
表示实际已开工但还未完工

图 10.2　以表格形式、条形框表示的甘特图

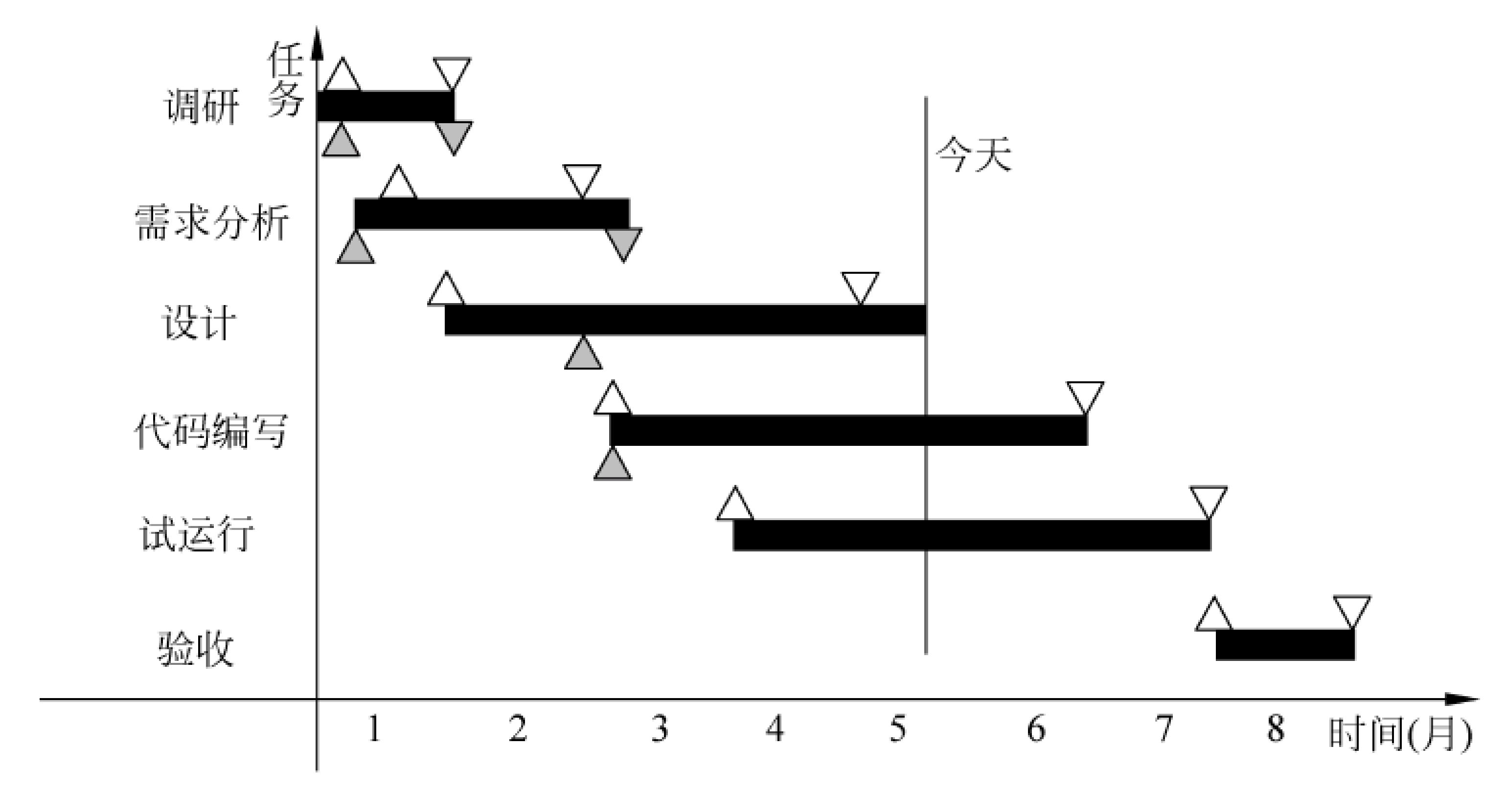
(2) 空心条框：表示实际完成情况，在一项任务完成时相应的空心条框也应绘制完成。起点表示实际开始时间，终点表示实际结束时间。

(3) 斜纹条框：表示实际已开工但还未完工，绘制到当前日期为止。起点表示实际开始时间，终点表示当前日期。

在图 10.2 中，相应的任务说明如下：任务“调研”已完成，且计划与实际情况相同；任务“需求分析”已完成，但实际工作比计划时间多，实际开工比计划早，完工却比计划晚；任务“设计”计划应完成，但实际未完成，且实际开始时间比计划开始时间晚；任务“代码编写”已开工，按计划还没完成，实际也未完成，计划开始时间与实际开始时间相同；任务“试运行”计划应已开工未完成，但实际未开工；任务“验收”计划未开工，实际也未开工。

形式 2：二维坐标、三角形框表示起止时间

用三角形框表示特定日期，向上三角形框表示开始时间，向下三角形框表示结束时间，计划时间和实际时间分别用空心三角形框和实心三角形框表示。一个任务只需要占用一行空间。用二维坐标、三角形框表示起止时间的甘特图如图 10.3 所示。任务的相关说明同图 10.2。



注释：△表示计划
▲表示实际

图 10.3　以二维坐标、三角形框表示的甘特图

10.2.2 持续时间压缩法

持续时间压缩法是一种数学分析方法，是在不改变项目范围的前提下，寻找缩短项目持续时间途径的方法。应急法和平行作业法都是持续时间压缩法。应急法是权衡成本和进度间的得失关系，以决定如何用最小增量成本达到最大量的时间压缩。应急法并不总是产生一个可行方案，且常常导致成本增加。平行作业法是使活动平行进行，有些活动通常要按前后顺序进行(例如，在设计完成前就开始编写程序)。平行作业法常常导致返工和风险增加。

一旦项目采用了合适的工作方法和工具，就可以简单地通过增加人员和加班时间来缩短进度，进行进度压缩。在进行进度压缩时存在一定的进度压缩和费用增长的关系，很多人提出不同的方法来估算进度压缩与费用增长的关系，这里介绍其中两种方法——线性关系法和进度压缩因子法。

1. 线性关系法

线性关系法是假设每个任务都存在一个“正常”进度和“压缩”进度，一个“正常”成本和“压缩”成本。如果任务在可压缩进度内，进度压缩与成本增长成正比。所以，可以通过计算任务单位进度压缩的成本来计算在压缩范围内进度压缩产生的压缩费用。

例如，图 10.4 是某项目的 PDM 网络图，如果 A、B、C、D 任务在可压缩范围内，进度压缩与成本增长呈线性正比关系。

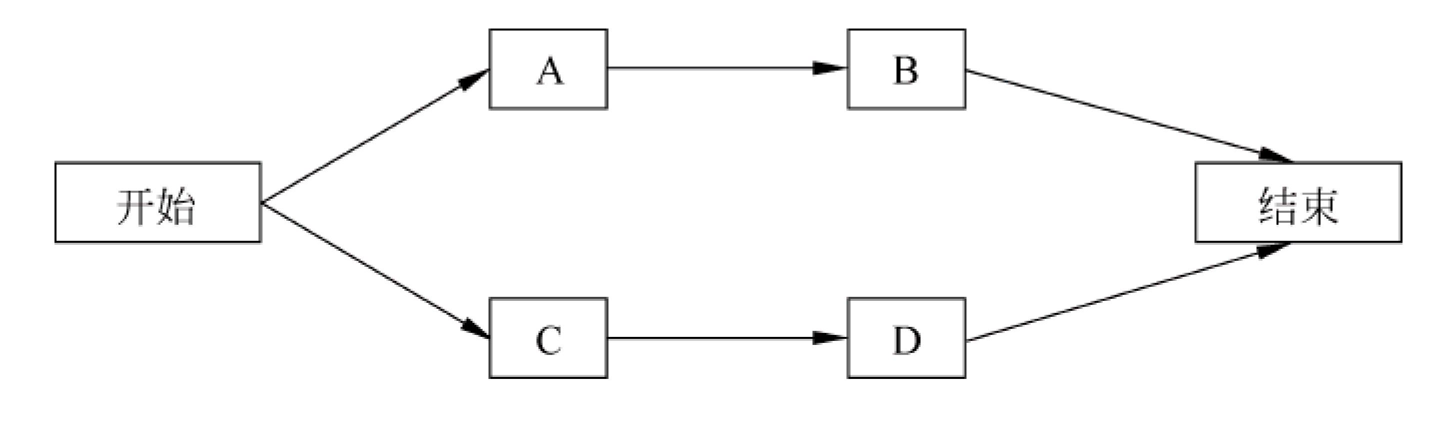


图 10.4 某项目的 PDM 网络图

表 10.1 给出了 A、B、C、D 各任务的历时估计、成本估计、压缩后的最短历时、压缩后的成本。从 PDM 网络图可知，目前项目的总工期为 18 周(A→B 路径为 16 周，C→D 路径为 18 周，故总工期为 18 周)，如果将工期分别压缩到 17 周、16 周、15 周，并且保证每个任务都在可压缩范围内，试求应该压缩哪些任务，并计算压缩后的总成本。分三步进行：

表 10.1 每个任务的历时和成本估计、压缩后的最短历时和成本

项目名称	A	B	C	D
历时估计	7 周	9 周	10 周	8 周
成本估计	5 万	8 万	4 万	3 万
压缩后的最短历时	5 周	6 周	9 周	6 周
压缩后的成本	6.2 万	11 万	4.5 万	4.2 万

(1) 首先计算 A、B、C、D 任务在可压缩范围内进度压缩与成本增长的线性正比关系，如表 10.2 所示。

表 10.2　每个任务的进度压缩与成本增长的线性正比关系

单位压缩成本	A	B	C	D
压缩成本(万/周)	0.6	1	0.5	0.6

(2) 如果将工期分别压缩到 17 周、16 周、15 周,并且保证每个任务在可压缩范围内,必须满足两个前提:

① A、B、C、D 任务必须在可压缩范围内。

② 保证压缩之后的成本最小。

(3) 根据上述两个条件,首先看可以压缩的任务,然后根据压缩后的情况计算总成本最小的情况,此情况为选择的压缩结果,如表 10.3 所示。如果希望总工期压缩到 17 周,可以压缩的任务有 C 和 D,但是根据表 10.2 知道压缩任务 C 的成本小(压缩任务 C 增加 0.5 万,压缩任务 D 增加 0.6 万元),故选择压缩任务 C。所以,项目压缩到 17 周后的总成本是 20.5 万元。同理,如果希望总工期压缩到 16 周,应该选择任务 D(任务 C 在可压缩的范围内是不可能再压缩的,否则压缩成本会非常高)。所以,项目压缩到 16 周后的总成本是 21.1 万元。如果希望总工期压缩到 15 周,应该选择压缩任务 A 和 D 各一周(这样的压缩成本最低)。所以,项目压缩到 15 周后的总成本是 22.3 万元。

表 10.3　压缩后的项目成本

完成周期(周)	可以压缩的任务	压缩的任务	成本计算(万元)	项目成本(万元)
18			5+8+4+3	20
17	C、D	C	20+0.5	20.5
16	C、D	D	20.5+0.6	21.1
15	A、B、C、D	A、D	21.1+0.6+0.6	22.3

2. 进度压缩因子法

进度压缩与费用上涨不是总能呈现正比关系,当进度被压缩到“正常”范围之外,工作量就会急剧增加,费用也会迅速上涨。而且,软件项目存在一个可能的最短进度,这个最短进度是不能突破的,如图 10.5 所示。在某些时候,增加更多的软件开发人员会减慢开发速度而不是加快开发速度。例如,一个人 5 天写 1000 行代码,5 个人 1 天不一定能写 1000 行代码,40 个人 1 小时不一定能写 1000 行代码。增加人员会存在更多的交流和管理时间。软件项目中存在这个最短的进度点。无论怎样努力工作,无论怎样聪明,无论怎样寻求创造性的解决办法,也无论组织多大的开发团队,都不能突破这个最短的进度点。

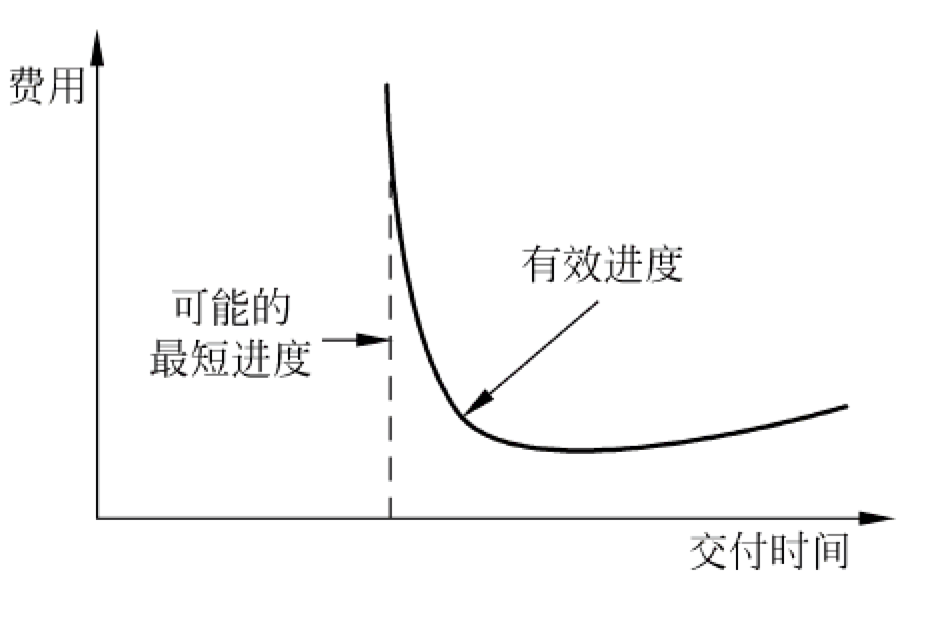


图 10.5　项目进度与费用的关系

由著名的软件度量专家 Charles Symons 提出的一种估算进度压缩费用的方法,被认为是精确度比较高的一种。公式为:

$$\text{进度压缩因子} = \text{期望进度} / \text{估算进度} \tag{10.1}$$

$$\text{压缩进度的工作量} = \text{估算工作量} / \text{进度压缩因子} \tag{10.2}$$

这种方法首先估算初始工作量和初始进度，然后将估算与期望的进度相结合，利用方程来计算进度压缩因子以及压缩进度后的工作量。例如，项目的初始估算进度是12个月，初始估算工作量是78人月，如果期望压缩到10个月，则进度压缩因子=10/12=0.83，压缩进度后的工作量=78/0.83=94人月。也就是说，进度缩短17%，增加21%的工作量。

很多研究表明：进度压缩因子不应该小于0.75，即进度压缩不应超过25%。

10.3 进度计划编制

进度计划通常是根据高级计划、总体计划制定，可细化为阶段计划和个人计划。它包括任务、资源和时间三部分内容。任务来源于工作分解结构和活动定义。要进行有效的进度控制，就要求必须有细致的、可执行的、可检查的、可控制的活动定义。任务的工作量要求适中。对于不成熟的项目和管理水平不高、资源能力不强的项目，任务的工作量不能太大，否则难以实现对项目的控制；对于成熟的项目和管理水平高、资源能力强的项目，任务的工作量就可以适当大一些。每项任务需要有明确的责任人、明确的工期。实际上，在软件项目管理水平不很高的情况下，要实现有效的进度控制，每项任务的工作量以不大于项目总体工作量的5%为宜，工期以不大于项目总工期的10%为宜。

10.3.1 任务的并行性

软件项目进度计划需要安排所有与该项目有关的活动，但在软件项目开发中，所有活动并不是完全独立、顺序进行的，有些活动是可以并行的。制定项目进度计划时，必须协调这些并行的任务并且组织这些工作，以使资源的利用率达到最优化，同时，还必须避免由于关键路径上的任务没有完成而导致整个项目推迟。任务的并行性示例如图10.6所示。

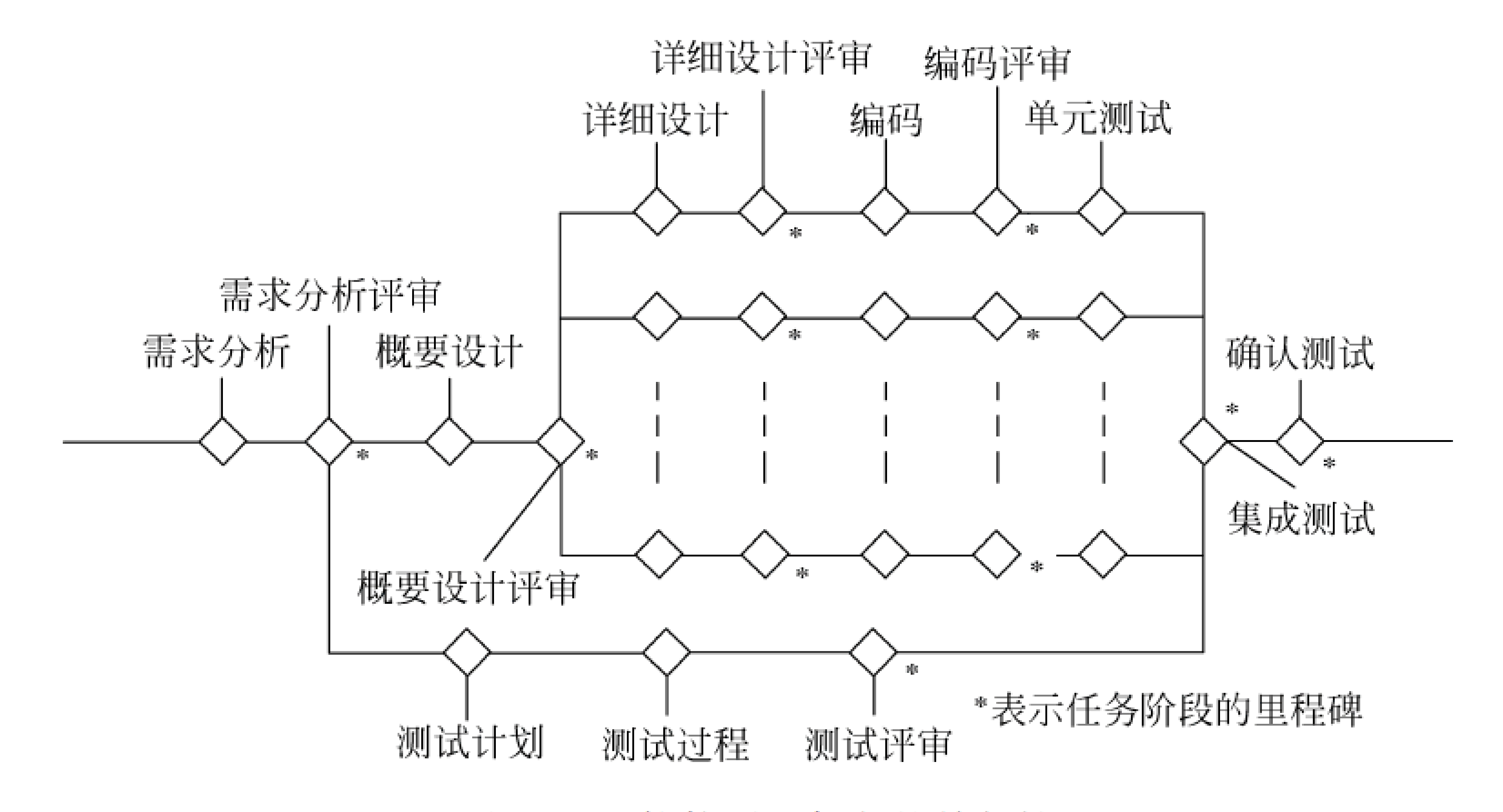


图10.6 软件项目任务的并行性

一般软件项目是很多人同时参加工作，因而项目中就会出现并行的情形。

软件工程过程设置了若干里程碑。里程碑为项目管理人员提供了考察项目进度的可靠

依据。当一个软件任务成功通过评审并产生相应文档后，就完成了一个里程碑。在软件项目过程中，首先要进行需求分析和评审，这为以后的并行工作奠定了基础。需求分析通过评审后，就可以并行开展概要设计和测试计划制定。概要设计通过评审，系统模块结构建立后，又可以并行地对各个模块进行详细设计、详细设计评审，编码、编码评审，单元测试。所有模块都调试通过后，组合在一起，进行集成测试。在软件交付前再做确认测试。

软件项目的并行性提出了一系列进度要求。因为并行任务同时发生，所以进度计划必须确定各个任务之间的从属关系、各个任务的先后次序和衔接以及各个任务的持续时间，以保证所有任务都能够按进度完成。

10.3.2 进度计划的表达形式

项目进度计划可以有多种表达形式，根据项目的具体情况选择和制定。常用的表达形式有以下几种。

1. 甘特图或条形图

甘特图或条形图是应用最广泛的进度计划表达形式，前文已述，在此不再重复。

2. 带有日历的项目网络图

在项目网络图中用节点（方框、椭圆或其他符号）表示任务，在节点内标出任务名称，如果任务较多，需要对每个任务进行编号；用带有单箭头的线段表示任务间的依赖关系（先后关系和并行关系）；在每个节点上方标明任务的开始时间和结束时间。示例如图 10.7 所示。

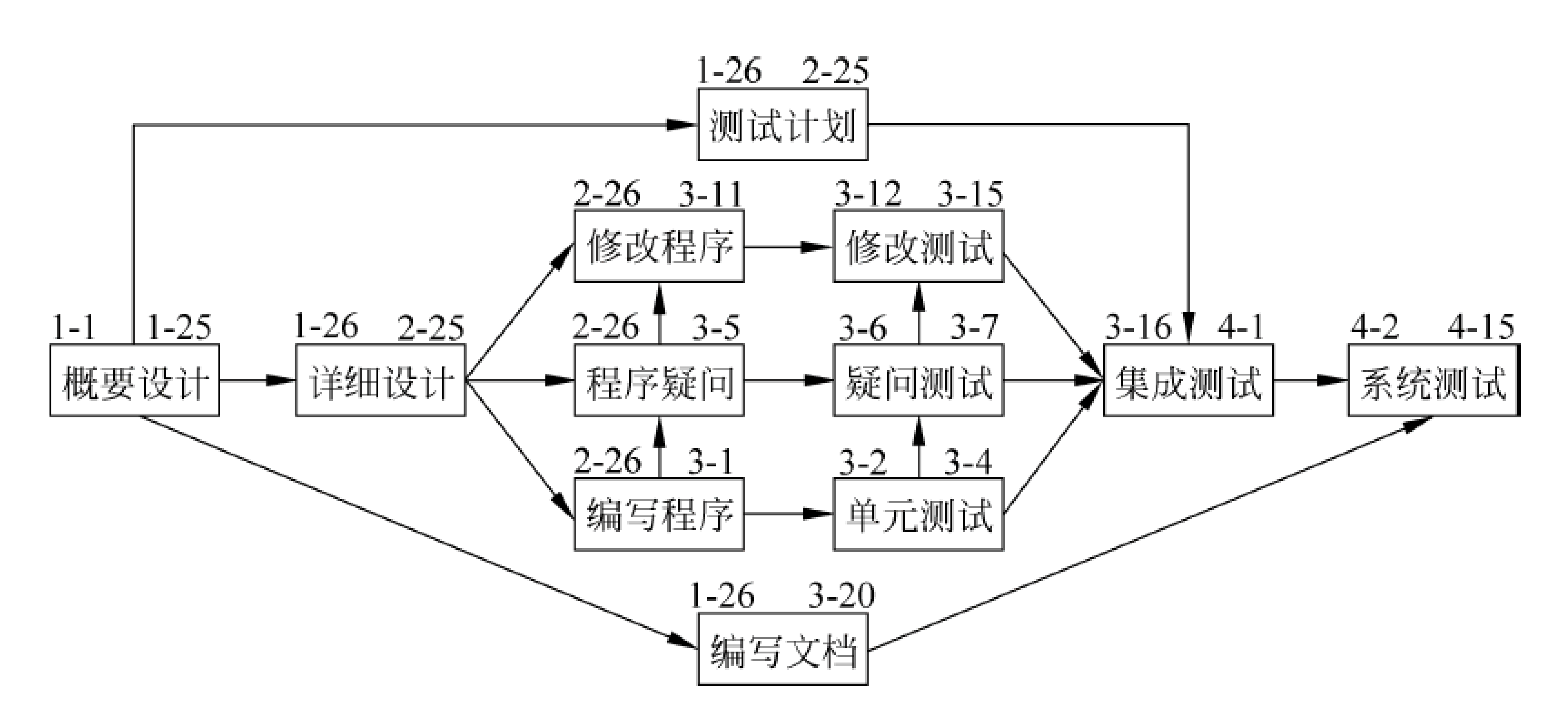


图 10.7 带有日历的项目网络图

3. 时间坐标网络图

在时间坐标网络图中，用带箭头的线段表示任务，线段的长短表示任务的持续时间，在线段的上方标出任务的名称，通过图中的坐标可以表明任务的开始时间和结束时间；用节点（方框、椭圆或其他符号）表示事件，即任务的开始和结束，在节点内写出事件的编号；用带虚线的箭头表示虚任务，实际并没有该任务，表示事件发生的先后关系。示例如图 10.8 所示，用字母 A、B、C、…表示任务。

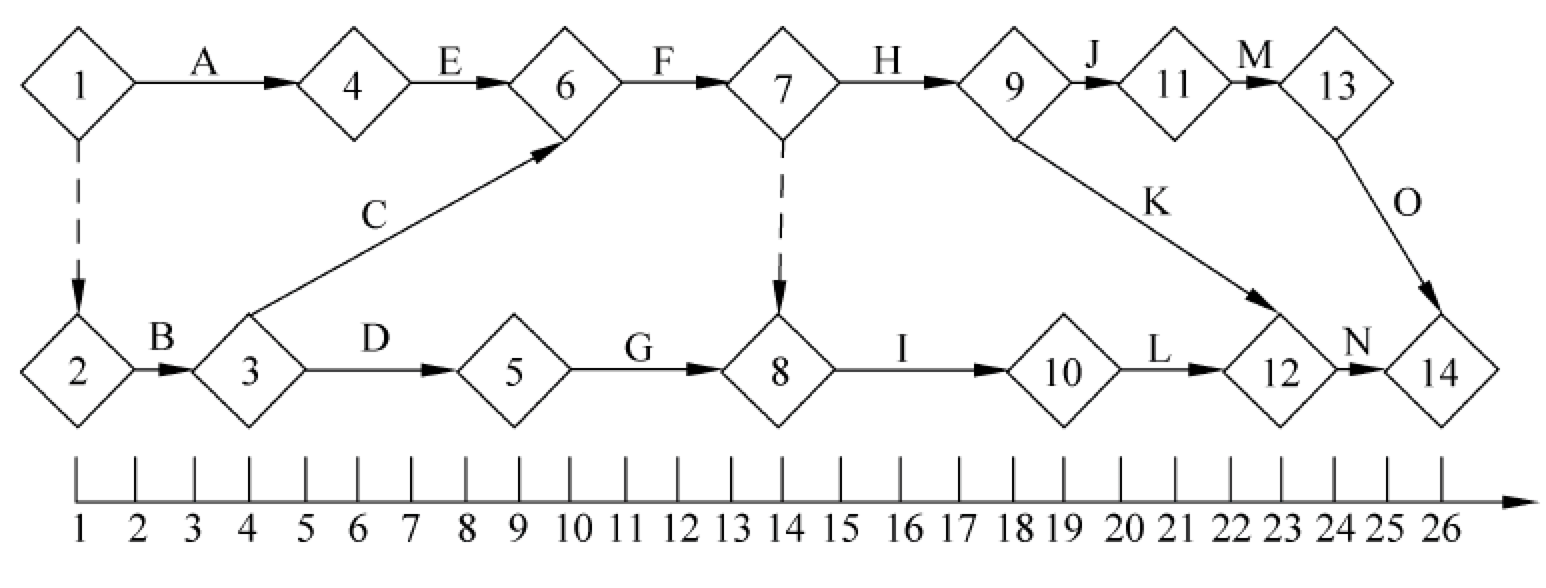


图 10.8 时间坐标网络图

4. 里程碑事件表

里程碑事件表说明了里程碑事件完成的大体时间，但不能说明任务的开始时间和结束时间，难以表达任务间错综复杂的关系。里程碑事件表通常在高层管理人员掌握项目的进展情况，或评审人员进行评审时使用。示例如表 10.4 所示。

表 10.4 里程碑事件表

序号	里程碑事件	时间											
		1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
1	合同签署	◆											
2	系统需求评审			◆									
3	软件需求评审					◆							
4	概要设计评审						◆						
5	详细设计评审								◆				
6	构件设计评审										◆		
7	集成测试评审											◆	
8	确认测试评审												◆

注：◆表示计划完成时间。

5. 项目任务列表

项目任务列表说明了项目的所有任务及各个任务的开始时间、结束时间、工期等，可将任务列表分配给所有的项目组成员，供开发时使用。某软件项目需求分析、概要设计两个阶段全部任务计划安排如表 10.5 所示。

表 10.5 项目任务列表

序号	任 务 名 称	开 始 时 间	结 束 时 间	工期(天)
0	全部任务	2009-3-1	2009-8-20	173
1	1. 需求分析	2009-3-1	2009-4-20	51
2	1.1 系统需求分析	2009-3-1	2009-3-25	25
3	1.2 软件需求分析	2009-3-11	2009-4-10	31
4	1.3 需求分析评审	2009-4-11	2009-4-20	41
5	2. 概要设计	2009-4-15	2009-6-10	57
6	2.1 模块结构设计	2009-4-15	2009-5-10	26
7	2.2 功能设计	2009-4-25	2009-5-20	26
8	2.3 接口设计	2009-5-1	2009-5-31	31
9	2.4 概要设计评审	2009-6-1	2009-6-10	10

10.3.3 进度计划编制过程

进度计划编制过程分为以下三个阶段。

第一阶段：前期准备

进度计划是在完成以下前期工作后进行的：

(1) 工作分解。将一个项目分解为更多的工作细目或子细目，使任务变得更小、更容易管理、更容易操作，是对需求的进一步细化，最后确定项目所有的任务范围。

(2) 建立网络图。通过网络图展示项目的所有活动，表示活动之间的依赖关系，表明任务将如何进行和以怎样的顺序进行。

(3) 估算活动持续时间。采用方法可以是计划评审技术、专家评定、经验类比、历史数据参考、德尔菲(Delphi)等。

(4) 为任务分配资源并进行平衡。为任务分配相应资源，然后对任务持续时间、开始日期、资源分配进行调整，从左到右平衡计划，保持各个任务之间的相互依赖关系。

(5) 确定管理支持性任务。管理支持性任务往往贯穿项目始终，具体指项目管理、项目会议等任务，以及需求、配置、测试、评审等其他支持性工作。

(6) 约束条件。项目完成受一定条件制约，这些条件在制定计划前必须考虑，包括资源配置、人员组成、时间限制等。

(7) 提前或滞后要求。关键活动要求按期完成，否则影响工作。但对于一些非关键活动，在时间安排上有一定的伸缩性，提前或推迟开工对整个项目可能更有益处。可根据资源运用情况适当向前或向后调整。

(8) 其他准备。包括确定项目组每个成员的可支配时间、假设条件、风险管理计划等。

第二阶段：计划编制

首先选择技术和方法，然后进行编制。常用的方法有以下几种：

(1) 关键路径法。这是一种运筹学方法，涉及的数学公式较多，本书没有讲述。

(2) 持续时间压缩法。

(3) 模板法。以前进度计划编制的结果为模板，在此基础上进行编制。

(4) 项目管理软件。运用计算机技术，通过项目管理软件进行计划编制。目前应用很多，并且会越来越广泛。现在应用最多的是 Microsoft Project。

第三阶段：输出结果

完成计划编制后，一般可得到如下结果：

(1) 项目进度计划。这是最重要的成果，表现形式是前文所述的甘特图或条形图、带有日历的项目网络图、时间坐标网络图、里程碑事件表、项目任务列表等。

(2) 制定项目进度计划依据说明，包括采用的技术、约束条件、限制条件、假设条件等，以及应用计划的详细说明。

(3) 更新的项目管理资料，包括工作分解结构、活动目录、持续时间估算、项目网络图、资源需求等。

10.4 进度计划控制

良好的计划是软件项目成功的基础,但在实际执行过程中,由于软件项目本身特点和一些不可预测因素,使得项目进展不能完全按照计划进行。为了确保项目取得成功,必须对项目计划执行过程进行追踪控制。

从软件项目实施结果来讲,能够在预定时间内达到预期工作目标,就可以说项目得到了有效的进度控制。

10.4.1 进度计划控制的难点

制定项目计划的过程也称为项目策划,在项目早期启动阶段,项目任务的目标、范围等还不是十分明确,但项目已经启动,项目组已经成立,就已经开始制定工作计划,此时的计划制定有很大的“预计”成分,相当于“策划”。在项目进入实质展开阶段,即合同签订、需求分析完成后,项目计划会是一个非常具体、严格考核的任务计划。这时,就不是策划而是计划了。进度计划控制的难点主要体现在以下几个方面:

(1) 软件项目的不确定性。

在项目策划时,要尽量让项目组成员估计自己的工期,使团队成员积极参与到项目中来,而且由于技术发展非常迅速,开发人员往往只对几个具体模块部分的工作有所了解。但是项目经理也不是完全被动的,可以通过积累项目管理数据,推动开发过程能力成熟度的提高,以便协助开发人员进行越来越准确的项目估计。

(2) 项目内容的隐蔽性与分散性。

软件项目内容通常不如其他项目那样具体,不易于集中收集整理,往往是分布在不同的人员手中。因此,计划要求以文本文档和图形文档结合的形式出现,文本主要记录项目的约束、限制、风险、资源、接口约定等方面的内容。对于进度、资源分解、职责分解、目标分解等,通过项目管理软件进行规划和管理,不要分散在文档的若干个地方,那样非常不利于同步修改。项目计划需要设计成“可检查”的文件,这要求任务划分具体到产品,如果存在有形的产品输出,要罗列出来。例如测试这一任务,不要简单分解为测试准备、测试执行,而是分解为测试环境搭建、测试方案编制、测试执行、测试报告编制等具体工作为好。

(3) 不能处理好计划与变化的关系。

在计划实施过程中,不能把计划固定化。在实际运作中,要对计划进行周期性维护。开发进度计划会受到很多方面影响,比如相关计划(质量保证计划、采购计划、测试计划、验收计划等)的影响、实际进度变动的影响、资源变动的影响、项目目标变动的影响,还有随着需求的逐渐明确引起的项目计划细化,如果这些变化发生后没有及时维护开发计划,开发计划与实际偏差会越来越大,最后变得没有价值,人们就会不再运用。所以在实际工作中,要有具体责任人和一套指导书对计划实施指导和维护。计划变更时,要保留旧版本,在总结阶段需要阅读旧版本信息,以便对项目过程变更历史作评价。总之,变化的计划才有生命力。

(4) 没养成按计划工作的习惯。

在实际工作中,执行项目计划常常遇到各种困难。一种情况是有些组织文化中有种观念认为计划是一种约束,只要大家努力向前赶就对了,没有必要自己捆住手脚;另一种情况是大家没有按照计划工作的习惯,计划虽然做好了,实际工作时还是我行我素,管理人员也没有维护计划的习惯,项目开始没多久,计划就被完全撂在一边;还有一种情况是资源不能保障,有时设备不到位,人员频繁抽调从事计划外工作,每天修改计划都来不及,只好放弃计划。最后一种情况常见于一些规模较小,还在"求生存阶段"的公司。

(5) 项目经理的权力过小。

在一个不十分规范的公司里,如果项目经理没有保证按计划执行的权力,计划就没有意义。维护项目经理权力的关键是激励和惩罚措施。

10.4.2　进度计划控制的手段

1. 执行信息收集

有效加强联系、沟通各方面信息是搞好进度控制的重要环节。为了控制好软件项目,需要对大量信息进行正规化收集和管理。这些信息包括谈话、信件、电话、电传、电子邮件、设计说明书等。收集和管理信息应遵循以下原则:

(1) 所有会议都应有正式记录。

(2) 文件、备忘录、会议记录中的事项必须说明由谁处理。

(3) 所有重要问题必须要有书面材料。

(4) 所有来往信件和电函都应编号存档。

(5) 保存完整档案。有些文件需分别存入不同的档案,如工程变动,应同时存入变动档案和技术档案。

2. 工作报告

工作报告由底向上逐层汇总而成,各成员汇报给小组长,小组长汇总后汇报给执行业务经理(如软件经理、评审经理、测试经理等),执行业务经理汇总后汇报给项目经理,项目经理汇总后再汇报给项目总监和总经理。

工作报告可采用定期和不定期两种形式。定期可按周、月、季、半年等,不定期应是项目中某个里程碑结束时。各成员应每周向小组长汇报,执行业务经理应每月向项目经理汇报,当里程碑结束时,项目经理应向总经理汇报。

项目经理汇报给总经理的项目进度报告内容可包括以下几个部分:

(1) 项目进度。近期完成的里程碑、取得的成绩、对项目有重大影响的事件等。

(2) 预算情况。以清晰、直观的图表反映项目预算情况,并对重大偏差做出解释。

(3) 存在问题。非常难解决的问题、对项目造成重大险情的事件等,同时提出要求高层管理人员支持的内容。

(4) 后续工作计划。对下期及后续工作简单说明,提供下一期的里程碑图表。

(5) 人员情况。人员变动情况,对项目起到突出作用的人员等。

3. 进度检查

在进度控制中，进度检查是最重要和最关键的工作，应进行定期或不定期检查。定期检查是指在预定的检查周期内执行的检查工作，检查周期由项目组根据软件项目的实际情况预先确定。对于时间跨度比较大的项目，可以相对长一些，如工期超过两年的项目，检查周期可以定为一个月。一般建议检查周期以不高于工期的5%～10%为宜。不定期检查可以在关键任务或里程碑任务计划完成时间进行。

进度检查工作可以分为四个步骤执行：

(1) 收集进度信息。可以有进度汇报和进度查验两种收集方法。进度查验就是项目经理采用直接检查方式，获取进展信息或验证的汇报信息。最好将两种方法结合使用。需要收集的信息包括任务执行状况和变更信息。任务执行状况包括任务实际开始和结束时间、当前任务完成程度等。变更信息包括资源变更、范围变更、与软件项目进度相关联的其他变更。

(2) 将项目进展信息与原计划进行比较。如果没有偏差，检查结束。如果存在偏差，执行下一步工作。

(3) 针对偏差寻求解决方案。如果出现进度偏差，针对偏差进行分析和研究，发现其中的问题，针对问题寻找解决方案。软件项目实施过程中出现进度偏差在所难免，实施进度控制就是要求能对偏差进行有效控制，提出相应的解决方案，使之有利于软件项目进展。通常采取的措施可以从控制进度偏差措施表中得到，如表10.6所示。

表10.6 项目进度偏差措施表

序号	方　法	适 用 条 件	不 利 因 素
1	加班赶进度	具备加班条件	增加成本，可能会降低资源的工作效率，引起副作用
2	增加资源	有可调用的资源，增加资源对项目进展有明显促进作用	增加成本，加大了沟通和任务安排上的难度，有时难以见效
3	协商解决问题	由于协调原因和配合方的工作不力造成了任务延期	无法解决现有的延期问题，只能对以后的工作起到作用，需要和其他方法结合起来使用
4	快速跟进	关键路径上的后续活动受延期活动的影响不大	可能会造成项目返工
5	调整进度计划，压缩后续关键路径工作工期	任务延期较严重，难以通过压缩任务来追赶项目进度	对后续工作的控制和实施工作要求较高
6	改进资源工作方法，提高资源工作能力	由于工作方法不合理或资源工作能力不足引起延期	不能立竿见影
7	优化进度计划，缩小项目范围，降低任务要求	项目进度的要求比范围和质量要求更高，原先制定的项目进度计划不尽合理，缩小项目范围、降低项目质量不会对项目后果产生较严重的影响	对项目整体工作和质量可能会产生影响

(4) 执行调整后的进度计划和解决方案。根据偏差处理决定,执行解决方案,调整项目进度计划。如果需要,通知项目干系人。当进度偏差比较大时,需要考虑缩小检查周期,以便更好地监视纠正,以保证项目按期完成。

4. 重要会议

软件项目开始以后,一般需要在各个关键时刻召开关键会议,以便能有效地追踪控制项目和发现解决问题。

关键会议的主要内容是总结上一阶段工作,分析出现的问题,针对问题提出新建议,并介绍下一阶段的主要任务和目标,使相关人员都能做到心中有数,明确工作方向。关键会议也是协调各不同学科、不同职能部门之间人员以及工作任务的重要手段。

除关键会议外,在软件项目进行的全过程中,应定期召开例会,如每周一次或每月一次。会上主要介绍项目进展情况,检查有无延期,提出存在问题和需要高层管理者协调的工作等,以便能及时发现并解决问题。

还有些非定期的特别会议,在必要时随时召开。如出现重大意外变化,有重要的分承包要进行,需要订购大型设备等。

由此可见,软件项目管理中的会议很多。需要项目负责人很好地把握全局,审时度势,使会议在关键的时间节点及时召开,以便有效地解决问题,确保项目顺利进展;否则,极易陷入会海之中,费时费力。

思考题

1. 软件项目进度计划的作用有哪些?
2. 软件项目计划管理过程分为哪些阶段?各阶段的主要工作是什么?
3. 在软件项目进度计划管理中要注意处理好哪些问题?
4. 甘特图有哪两种形式?如何绘制甘特图?
5. 持续时间压缩法的线性关系法如何应用?
6. 持续时间压缩法的进度压缩因子法如何应用?
7. 如何理解软件项目开发的任务并行性?
8. 项目进度计划有哪些表达形式?每种形式如何应用?
9. 进度计划编制过程中的前期准备工作有哪些?
10. 进度计划编制过程中的输出结果有哪些?
11. 进度计划控制的难点包括哪些方面?如何理解这些难点?
12. 进度计划控制的手段具体包括哪些?

第11章 质量管理

软件项目质量管理是贯穿整个软件生命周期的重要工作，是软件项目顺利实施并成功完成的可靠保证。随着软件开发技术的发展和信息技术的广泛应用，软件项目质量管理越来越受到重视。实现软件项目质量管理与国际标准接轨，加强软件管理，改善软件开发过程，提高软件质量，已成为软件行业面临的巨大难题。

不断提高软件质量是软件开发的永久性工作。人们对软件产品的依赖越来越强，软件质量问题带来的危害也越来越严重；人们对软件质量的要求越来越高，对质量控制和质量管理也就越来越重视。本章就软件项目质量管理相关的质量控制、质量评审、质量度量、质量管理体系等内容进行讲述。

11.1 软件质量与软件质量管理

11.1.1 软件质量

1. 软件质量定义

软件质量与传统意义的质量概念并无本质区别，软件质量具有一般质量概念的共性，但也有固有特性，就是产品或服务满足用户的程度。

概括地说，软件质量就是“软件与明确和隐含定义的需求相一致的程度”。具体地说，软件质量是软件符合明确叙述的功能和性能需求、文档中明确描述的开发标准以及所有专业开发的软件都应具有的隐含特征的程度。软件质量定义强调以下三点：

(1) 软件需求是度量软件质量的基础，与需求不一致就是质量不高。

(2) 指定的标准定义了一组指导软件开发的准则，如果没有遵守这些准则，几乎肯定会导致质量不高。

(3) 通常有一组没有显式描述的隐含需求(如期望软件容易维护)。如果软件满足明确描述的需求，但不满足隐含的需求，那么软件的质量仍然值得怀疑。

2. 软件质量模型

早在 1976 年，由 Boehm 等人提出软件质量模型的分层方案。1979 年，McCall 等人改进 Boehm 质量模型又提出了一种软件质量模型。其层次式框架如图 11.1 所示。软件质量模型中的质量概念基于 11 个特性，而这 11 个特性分别面向软件产品的运行、修正、转移。

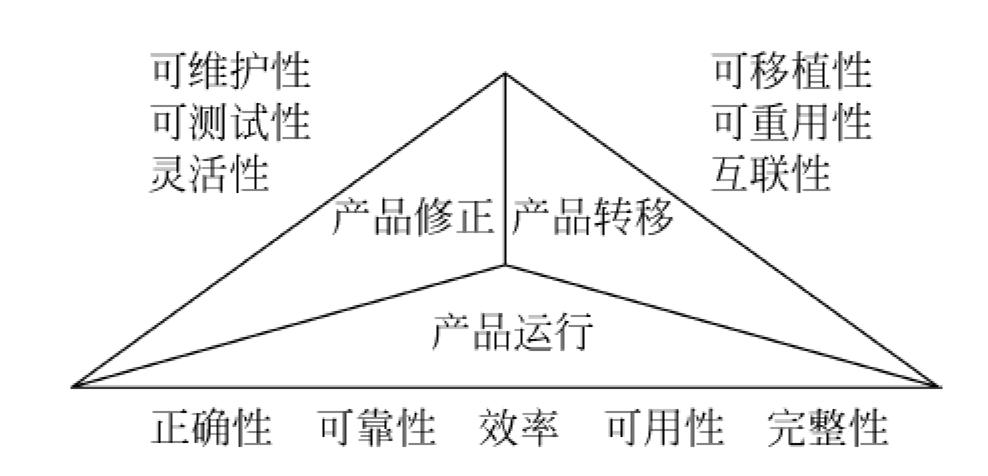


图 11.1 McCall 软件质量模型层次式框架

McCall 等人认为,特性是软件质量的反映,软件属性可用做评价准则,定量化地度量软件属性可知软件质量的优劣。

McCall 等人的质量特性属性描述如表 11.1 所示。

表 11.1 软件质量特性属性描述

序号	属性名称	描 述
1	正确性	在特定环境下,软件满足设计规格说明及用户预期目标的程度。要求软件本身没有错误
2	可靠性	软件按照设计要求,在规定的时间和条件下不出故障、持续运行的程度
3	效率	为了完成预定功能,软件系统所需计算机资源的多少
4	完整性	为某一目的而保护数据,避免受到偶然或有意的破坏、改动、遗失的能力
5	可用性	对于一个软件系统,用户学习、使用软件以及为程序准备输入和解释输出所需工作量的大小
6	可维护性	为满足用户的新要求,或环境发生了变化,或运行中发现了新的错误时,对一个已投入运行的软件进行相应诊断和修改所需工作量的大小
7	可测试性	测试软件以确保能够执行预定功能所需工作量的大小
8	灵活性	修改或改进一个已投入运行的软件所需工作量的大小
9	可移植性	将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小
10	可重用性	一个软件(或软件部件)能再次用于其他应用(该应用的功能与此软件或软件部件完成的功能有关)的程度
11	互联性	联结一个软件和其他系统所需工作量的大小,又称相互操作性。如果这个软件要联网,或与其他系统通信,或要把其他系统纳入到本软件的控制之下,必须有系统间的接口,使之可以联结

11.1.2 软件质量管理

软件质量管理的主要内容包括保证软件满足目标需要的过程,涵盖了软件质量方面的指挥和控制活动,通常指制定软件质量目标以及进行质量策划、质量计划、质量保证、质量控制、质量改进等。

(1) 质量策划。质量策划是软件质量管理的一部分,致力于制定软件目标并规定必要的运行过程和相关资源,以实现软件质量目标。软件质量目标是软件质量管理追求的目的。

(2) 质量计划。结合公司的质量方针、产品描述以及质量标准和规则,通过收益、成本分析和流程设计等手段制定实施策略,为质量小组成员有效工作提供指南,为项目相关人员

了解在项目进行中实施质量保证和控制提供依据，为确保项目质量得到保障提供基础。

(3) 质量保证。质量保证是贯穿整个项目生命周期的有计划和有系统的活动，经常性地针对整个项目质量计划的执行情况进行评估、检查与改进等工作，向管理者、用户或其他方提供信任，确保项目质量与计划保持一致。

(4) 质量控制。质量控制是对阶段性成果进行检测、验证，为质量保证提供参考依据，是一个 PDCA(Plan——计划、Do——执行、Check——检查、Action——处理)循环过程。

(5) 质量改进。在充分运用质量保证、质量控制等方法的基础上，寻求改进质量的方法，进一步提高软件质量。

11.2 软件质量策划

质量策划包括识别和确定必要的作业过程，配置所需的人力和物力资源，以确保达到预期质量目标所进行的周密考虑和统筹安排过程。项目质量策划是保证项目成功的过程之一。

在国际标准 ISO 9000—2000 中将质量策划定义为：质量策划是质量管理的一部分，致力于设定质量目标并规定必要的运行过程和相关资源以实现质量目标。具体地说，就是根据项目内外部环境制定质量目标和计划，同时为保证目标实现，规定相关资源的配置。

质量策划过程一般包括四个方面的工作：

(1) 收集资料。策划必须建立在事实基础上，要明确和收集制定质量策划时所需的资料和数据，主要包括以往类似项目的质量策划资料，以及在执行和处理现场情况时总结的经验教训资料、数据对比资料、质量策划变更记录资料等，其他还包括项目组目前可以支配的资源、项目相关方已完成的工作、项目当前状况、项目投资人对项目未来的期望等。

(2) 进行相关内容的策划。主要包括以下内容：

① 产品质量策划，包括对老产品改进和新产品开发进行筹划，确定产品的质量特性、质量目标和要求，规定相应的作业过程和相关资源以实现质量目标。

② 质量管理和作业策划，包括确定项目所涉及的质量管理体系的过程内容，明确作业内容，规定相应的管理过程和相关资源，以达到控制要求，实现管理目标。

③ 编制质量计划。为满足质量要求，根据组织自身条件开展一系列的筹划和组织活动，提出明确的质量目标和要求，制定相应的质量管理过程和资源文件，包括质量责任、质量活动顺序等。

(3) 学习和使用质量策划的科学方法。进行质量策划时必须依靠科学方法，包括工具、技术、知识和经验等。

(4) 写出质量策划书和有关辅助文件。

11.3 软件质量计划

质量计划是软件项目整体计划的组成部分之一，制定软件质量计划的目的是确保软件质量标准在项目开发和维护过程中得到执行，使项目按期完成。质量计划是指导软件项目

整体计划的纲领性文件，其他计划应服从于质量计划的要求。

项目承办单位(或软件开发单位)中负责软件质量的机构或个人，必须制定软件质量计划，并由项目委托单位和项目承办单位的代表共同签字、批准。软件质量计划规定在项目中采用软件质量保证的措施、方法和步骤。

软件质量计划的内容如下。

1. 管理

描述负责软件质量的机构、任务及其有关的职责。

(1) 机构。描述与软件质量有关的机构组成，还必须清楚地描述来自项目委托单位、项目承办单位、软件开发单位和用户中负责软件质量的各个成员在机构中的相互关系。

(2) 任务。描述计划涉及的软件生命周期中有关阶段的任务，特别是重点描述这些阶段进行的软件质量保证活动。

(3) 职责。指明软件质量计划中每一项任务的负责单位或成员的责任。

2. 文档

列出在软件开发、验证、确认、使用与维护等阶段需要编制的文档，并描述对文档进行评审与检查的准则。

(1) 基本文档。为确保软件满足需求，至少需要的基本文档包括软件需求规格说明、软件(结构)设计说明、测试计划与测试报告、软件验证与确认计划、软件验证和确认报告等。

(2) 用户文档。需要编制的用户文档，包括用户手册、操作手册等。

(3) 其他文档。其他文档包括项目开发计划(可包括软件配置管理计划，必要时软件配置管理计划也可单列)、项目进展报表、项目开发各阶段的评审报表、项目总结报告等。

3. 评审和检查

规定要进行的技术和管理两方面的评审和检查工作，并编制或引用有关的评审和检查规程以及通过与否的技术准则。至少要进行下列评审和检查工作：

(1) 软件需求规格评审。在软件需求分析阶段结束后进行评审，确保在软件需求规格说明中规定的各项需求的合理性。

(2) 系统/子系统设计评审。在系统/子系统设计结束后进行评审，评价软件在总体结构、外部接口、主要部件功能分配、全局数据结构以及各主要部件之间的接口等方面的合理性。

(3) 软件设计评审。在软件设计结束后进行评审，评价软件(结构)设计说明中所描述的软件设计，主要是在功能、算法和过程描述等方面的合理性。

(4) 软件验证与确认计划评审。在制定软件验证与确认计划之后进行评审，以评价验证与确认方法的合理性与完整性。

(5) 功能检查。软件发行前，对功能进行检查，确认已经满足软件需求规格说明中规定的所有需求。

(6) 物理检查。验收软件前，对软件进行物理检查，验证程序和文档是否已经一致并已做好了交付准备。

(7) 综合检查。软件验收时，允许用户或用户委托的专家对验收的软件进行设计抽样

综合检查,验证代码和设计文档的一致性、接口规格说明之间的一致性、设计实现和功能需求之间的一致性、功能需求和测试描述的一致性。

(8) 管理评审。对计划的执行情况定期(或按阶段)进行评审,由独立于被评审单位的机构或授权的第三方主持进行。

4. 评审和审核

(1) 过程评审。描述对项目进行过程评审的方法和依据,并在样式如表 11.2 的表格中列出项目定义的过程以及相应的过程评审。

表 11.2 过程评审与产品审核记录表

阶 段	项目定义的过程	工作产品	质量记录	评审/审核活动

(2) 工作产品审核。描述进行产品审核的方法和依据,并在样式如表 11.2 的表格中列出项目过程应产生的工作产品和质量记录,以及需要由 SQA 负责人审核的工作产品和相应的产品审核活动。

(3) 不符合问题的解决。描述评审中出现的不符合问题的解决方法。

5. 其他

(1) 软件配置管理。编制有关软件配置管理条款,或单独制定文档。

(2) 工具、技术和方法。指明用以支持特定软件项目质量工作的工具、技术和方法。

(3) 媒体控制。指出保护计算机程序物理媒体的方法和设施,避免非法存取、意外损坏或自然老化。

(4) 对供货单位的控制。供货单位包括项目承办单位、软件销售单位、软件开发单位等。规定对供货单位进行控制的规程,保证供货单位提供的软件能满足规定的需求。

(5) 记录的收集、维护和保存。指明需要保存的软件质量活动记录,指出用于汇总、保护和维护这些记录的方法和设施,并指明保存期限。

(6) 日程表。列出项目质量活动日程表,并确保日程表与项目开发计划以及配置管理计划保持一致。日程表的格式如表 11.3 所示。

表 11.3 软件质量活动日程表

阶 段	活 动	日 期

11.4 软件质量保证

软件质量保证(Software Quality Assurance,SQA)是指确定、达到和维护所需要的软件质量而进行的有计划、有组织的管理活动。其目标是以独立审查方式,从第三方的角度监

控软件开发任务的执行，就软件项目是否正确遵循已制定的计划、标准和规程，给开发人员和管理层提供反映产品和过程质量的信息和数据，提高项目透明度，同时辅助软件工程组取得高质量的软件产品。

11.4.1 质量保证活动

软件质量保证活动很多，通常包括SQA计划、需求管理、静态测试、动态测试、过程管理、质量管理等。

1. SQA目标与活动的对应关系

结合软件质量目标，可以获得与SQA相关的各种SQA活动的对应关系，如图11.2所示。通过对应关系，在给定的目标下，可以迅速找出相对应的策略，确保SQA活动的实时性、高效性。

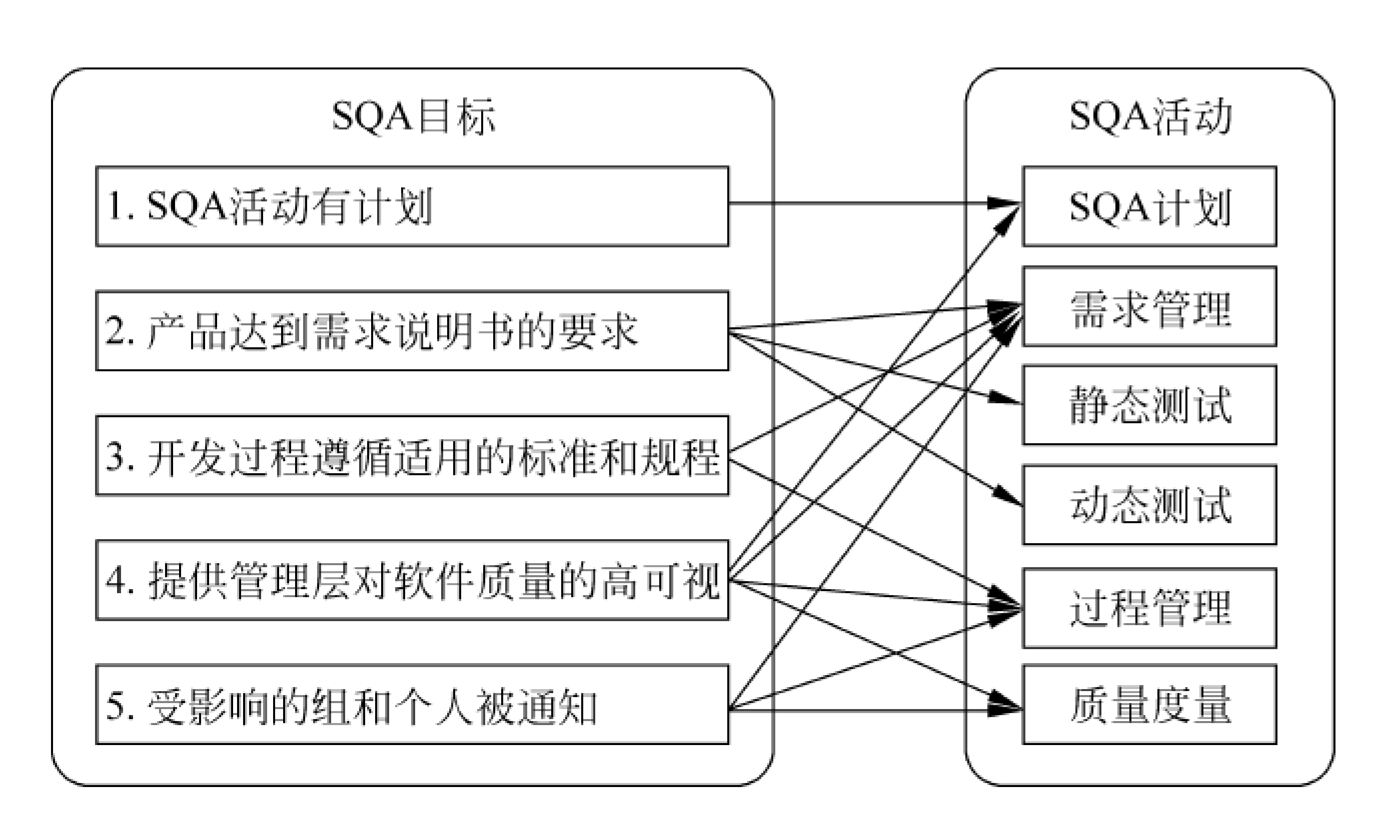


图11.2 SQA目标与活动的对应关系

2. SQA活动分布

以瀑布模型为基础，将一般的软件开发过程划分为计划、需求分析、软件设计、编码实现及软件测试等阶段，最后完成提交。质量保证实施通常基于以上软件过程。图11.3显示了在软件生命周期中理想情况下的SQA活动分布。通过估算某个阶段SQA活动的工作量，更好地调整资源分配，缩短关键路径时间。

3. SQA活动描述

对于各活动描述如下：

(1) SQA计划。这是实现目标1和目标4的基础，在SQA活动之前进行，属于软件过程的计划阶段。SQA计划前需要深入了解项目任务及SQA目标。前期收集分析项目资料并做出合理估算，有利于降低风险，减少计划变更。

(2) 需求管理。需求的频繁变更引起计划变更、文档陈旧、交流混乱、延长工期、降低软件质量。特别需要注意两个问题：一是明确开发人员与SQA人员在需求管理方面的职责，保证需求被各方理解并以标准化文档的形式保存，SQA要对需求文档进行审阅；二是需求变更时，SQA要保证需求变更遵照一定的规范。

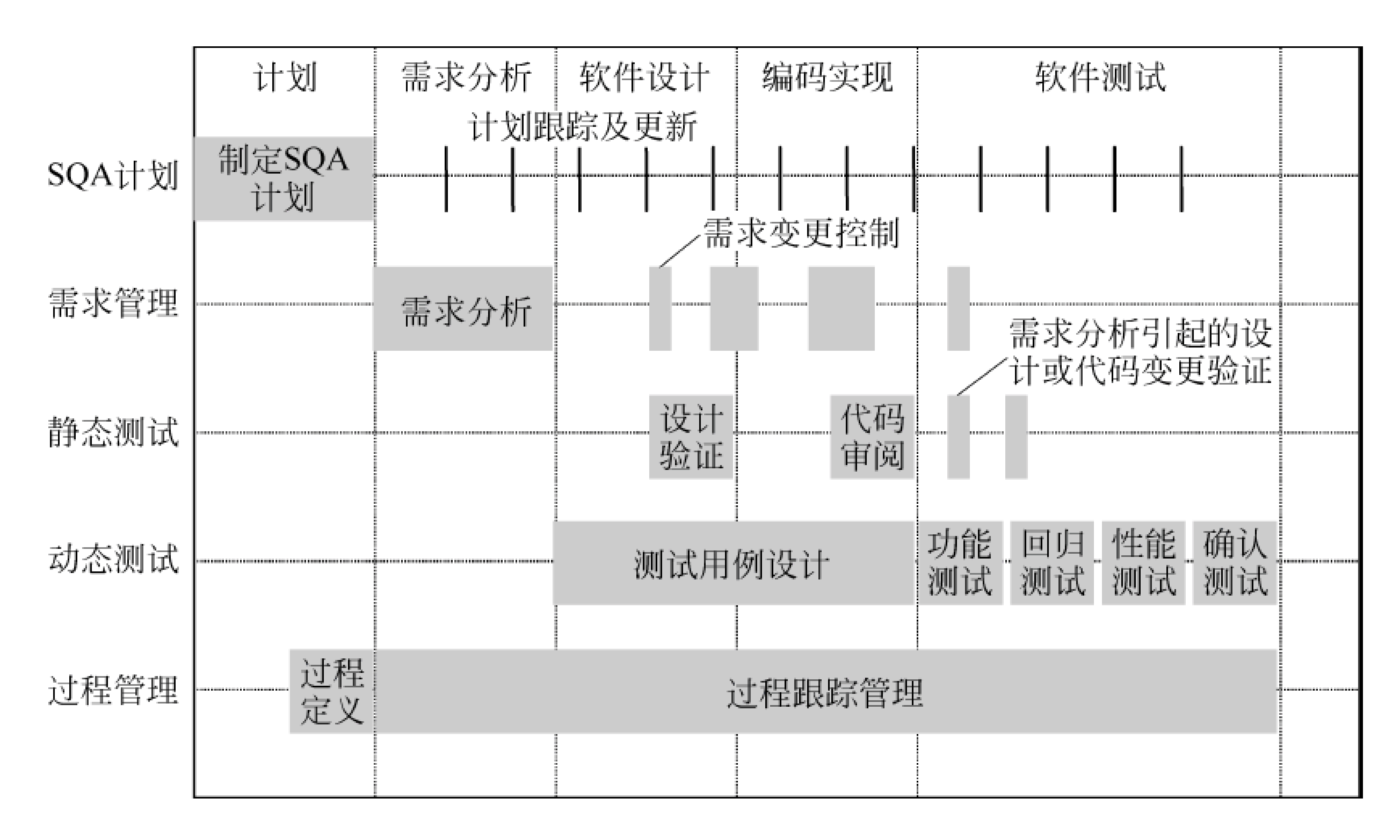


图 11.3 SQA 活动分布

(3) 静态测试。静态测试包括开发文档审阅、设计架构验证、代码评审等，是实现目标 2 的途径之一，主要发生在设计及代码实现阶段后期。静态测试提前到开发前期，有利于及早发现软件缺陷，节省修复缺陷成本。

(4) 动态测试。这是实现目标 2 的最重要途径，一般包括功能测试、回归测试、性能测试与确认测试等。这些测试在时间上顺序执行，并在代码完成并通过审阅后进行；测试用例设计与测试数据准备在需求分析完成之后、代码提交测试之前完成。

(5) 过程管理。过程管理是指在软件开发过程中对需要遵循的规范进行管理。为了便于开发人员与 SQA 部门沟通，在项目初始阶段制定规范，例如代码提交 SQA 测试时必须达到的标准、SQA 测试完成并提交给软件用户时必须达到的标准、缺陷跟踪管理、软件版本发布管理等。这些规范在 SQA 计划中被详细定义，与软件用户达成共识，贯穿整个软件过程。

(6) 其他 SQA 活动。SQA 定期报告、质量度量矩阵图、缺陷统计数据等方法是实现目标 4 的主要途径。

11.4.2 质量保证关键技术

软件质量保证是开发高质量软件的一系列活动过程。这个过程是多维的、复杂的，需要从技术、工具、手段、管理等多方面入手。质量保证包括以下一些关键技术。

1. 软件复用技术

大多数工程项目都尽可能采用复用的零部件，就如同没有哪位计算机硬件工程师会从电阻、电容和集成电路开始设计一样。但很多软件开发工作一切从头开始，是软件开发效率低、成本高、质量差的根本原因。近年来，软件复用技术研究活跃，取得了一些成果，成为解决软件危机的途径之一。

软件复用的宗旨是使软件开发工作速度快、费用省、质量好，复用的本质是共享复用构

件系统。被复用的构件应是成熟的、可靠的、通用的，初次开发时已经通过了审查和评审，构件的代码均经过单元测试、系统测试和多次现场测试。如果软件大部分都是由能复用的构件组成，质量将会大大提高。

2. 新的软件开发技术

随着复用构件技术的发展，新的软件开发技术层出不穷。以“构件”作为关键，复用大粒度的“对象”，链接与嵌入外系统的对象，称为面向对象的开发技术。这种软件开发技术灵活快捷，开发出的软件美观易用，深受软件开发人员和用户的喜爱。这些新技术包括微软公司的 Visual 系列、ActiveX、OLE（Objects Linking and Embedding，对象的链接与嵌入）、Sun 公司的 Java、OMG 公司的 CORBA（Common Object Request Broker Architecture，公用对象请求代理程序体系结构）、IDL（Interface Definition Language，接口定义语言）等。

在新的软件开发技术支持下，大量的输入输出界面只需要构件组装、多媒体对象灵活嵌入，程序代码量减少，代码出错率降低，测试投入下降，开发效率提高，软件质量得到保证。另外，利用结构化分析及设计技术和软件的模块化结构，也是降低成本、提高系统可维护性、提高软件质量的有效方法。

3. 容错技术

无论使用多么高明的避开错误技术，也无法做到完美无缺，这就需要采用容错技术，以使错误发生时不影响系统特性，对用户的影响限制在允许范围内。具有容错功能的软件在一定程度上对自身错误有屏蔽能力，也能从错误状态自动恢复到正常状态，发生错误时仍能完成预期功能。

实现容错技术的主要手段是冗余，包括以下三种：

(1) 结构冗余。结构冗余是由多人同时用不同的方式开发功能相同的模块，运行时通过表决和比较进行选择，借此来屏蔽系统中出现的错误。

(2) 信息冗余。信息冗余是以检测和纠正信息在运算或传输中的错误为目的而外加的一部分信息，例如奇偶码、定重码、循环码等冗余码形式，发现甚至纠正通信和计算机系统中的错误。

(3) 时间冗余。时间冗余是以重复执行指令或程序来消除瞬时错误带来的影响，当有错误恢复请求信号时，重复执行该指令，如果重复执行仍然无效，则发出中断，转入错误处理程序。

11.5 软件质量控制

高质量的软件离不开有效的管理和控制。质量控制是一个常规过程，通过度量实际的质量性能并与标准比较，当出现差异时采取行动。软件质量控制是一系列验证活动，采取有效措施监控软件开发过程，在软件开发过程的任何一点评估开发产品在技术上是否符合为该阶段制定的规约。通过一系列验证活动，对发生的错误或者偏差及时纠正，而不是直到最后才发现问题，以致无法弥补。质量计划确定后，就要按照质量管理体系，实施有效的质量控制。质量控制分为监测和控制两个阶段。监测是收集、记录和汇报有关项目质量信息；

控制是通过质量监测提供的数据进行控制，确保项目质量与计划保持一致。

11.5.1 质量控制模型

PDCA 循环又叫戴明环，是管理学的一个通用模型，最早由休哈特（Walter A. Shewhart）于 1930 年构想，后来被美国质量管理专家戴明（Edwards Deming）博士于 1950 年再度挖掘出来，并加以广泛宣传和运用于持续改善产品质量过程中，成为全面质量管理遵循的科学程序。全面质量管理活动的过程就是质量计划的制定和组织实现的过程，这个过程就是按照 PDCA 循环进行的。将 PDCA 循环用于质量控制的质量控制模型如图 11.4 所示。

1. PDCA 循环上升过程

PDCA 质量控制方法是循环的、闭合的，同时也是螺旋式上升的，经过 PDCA 的多次循环和升华，使得项目质量始终处于受控状态。PDCA 循环上升示意图如图 11.5 所示。

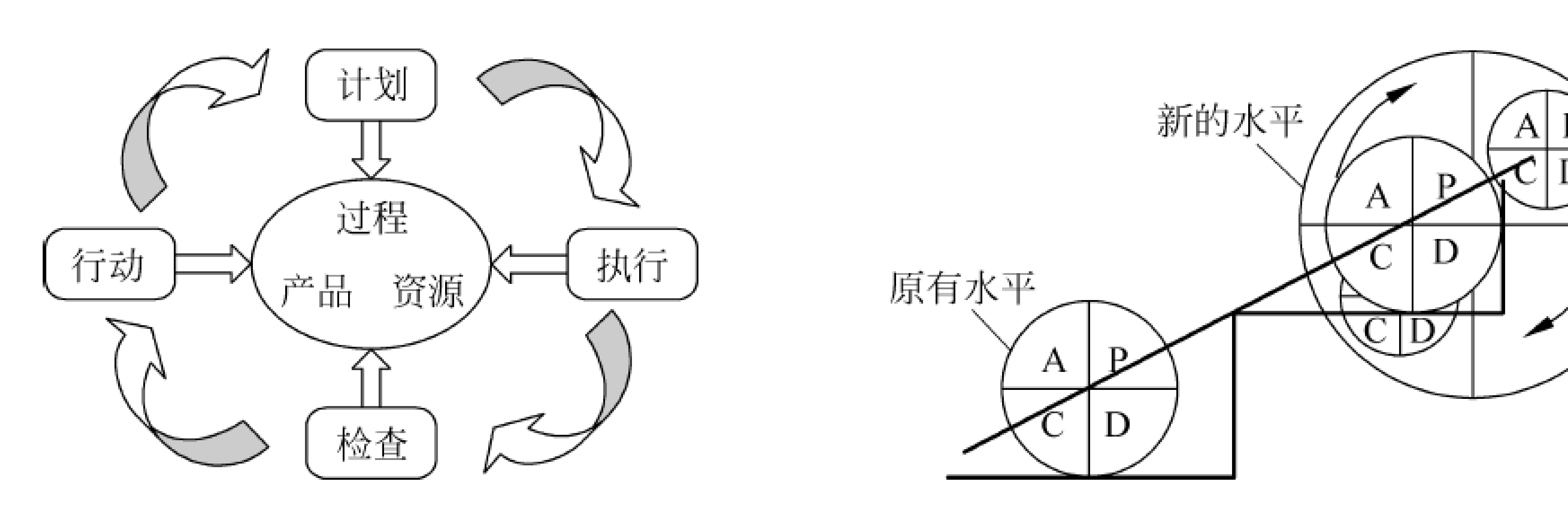


图 11.4 质量控制模型　　图 11.5 PDCA 循环上升示意图

对图 11.5 的说明如下：

（1）各级质量管理都是一个 PDCA 循环，形成一个大环套小环、一环扣一环、互相制约、互为补充的有机整体。在 PDCA 循环中，上一级循环是下一级循环的依据，下一级循环是上一级循环的落实和具体化。

（2）每个 PDCA 循环不是在原地周而复始地运转，而是都有新的目标和内容，这意味着质量管理经过一次循环解决一批问题，质量水平有了新的提高。

2. PDCA 的四个过程八个阶段

PDCA 包括四个过程八个阶段，如图 11.6 所示。

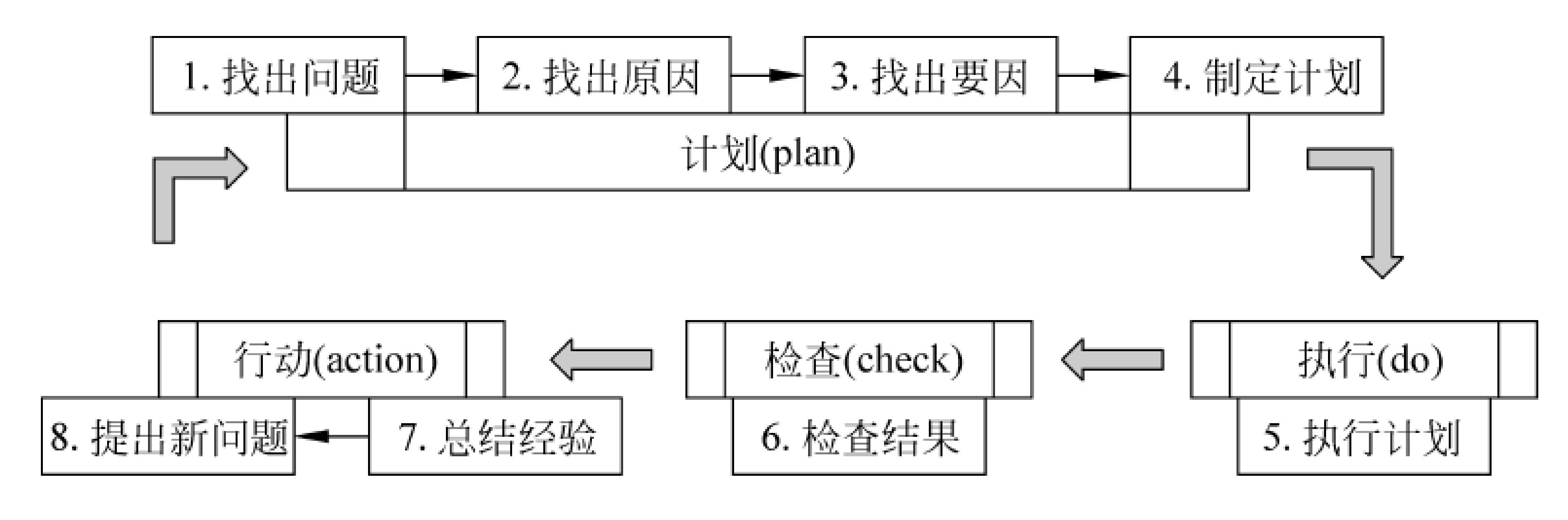


图 11.6 PDCA 的四个过程八个阶段

(1) 计划(plan)。分析现状,发现问题,找出原因,制定相应的质量方针、目标、计划和原则。该阶段包括四个阶段,即找出问题、找出原因、找出要因、制定计划。

(2) 执行(do)。根据计划实施,执行计划中规定的各项活动。该阶段包括一个阶段,即执行计划。

(3) 检查(check)。对执行的结果进行检查、审核和评估,收集数据并进行分析,度量工作的质量,发现存在的问题。该阶段包括一个阶段,即检查结果。

(4) 行动(action)。针对检查中发现的问题,采取相应的改进措施纠正偏差。总结成功经验,吸取失败教训,形成标准和规范指导以后的工作,通过行动提高并升华。该阶段包括两个阶段,即总结经验、提出新问题。

11.5.2 质量控制的方法与技术

软件质量控制有因果图、控制图、流程图、直方图、Pareto 图、趋势图、散点图七种方法与技术,被业界称为质量控制七工具。这里介绍其中的四种。

1. 因果图

日本著名质量管理专家石川馨(Kaoru Ishikawa)发明的因果图又称为石川图或鱼刺图,直观地显示出各项因素如何与各种潜在问题或结果联系起来。利用因果图可以将在产品后端发现的质量问题一直追溯到负有生产责任的人员或过程,从生产源头找出质量原因,真正获得质量的改进与提高。因果图示例如图 11.7 所示。

2. Pareto 图

意大利著名经济学家帕累托(Pareto)提出了"关键的少数和无关紧要的多数之间的关系",有时称为二八原理,即 80%的问题经常是由于 20%的原因引起的。朱兰把这一规则引进产品质量管理,以确认造成系统质量问题的诸多因素中最为重要的几个因素。Pareto 图又称为排列图或主次因素分析图,是用于帮助确认问题和对问题进行排序的一种常用的统计分析工具,其基本格式如图 11.8 所示。

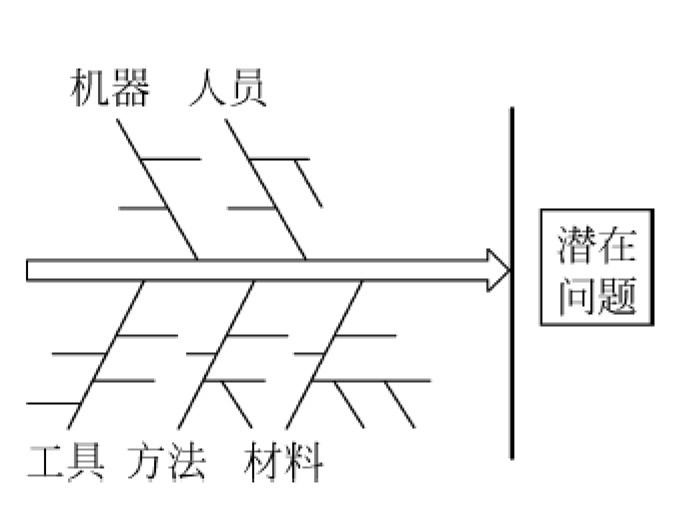


图 11.7 因果图示例

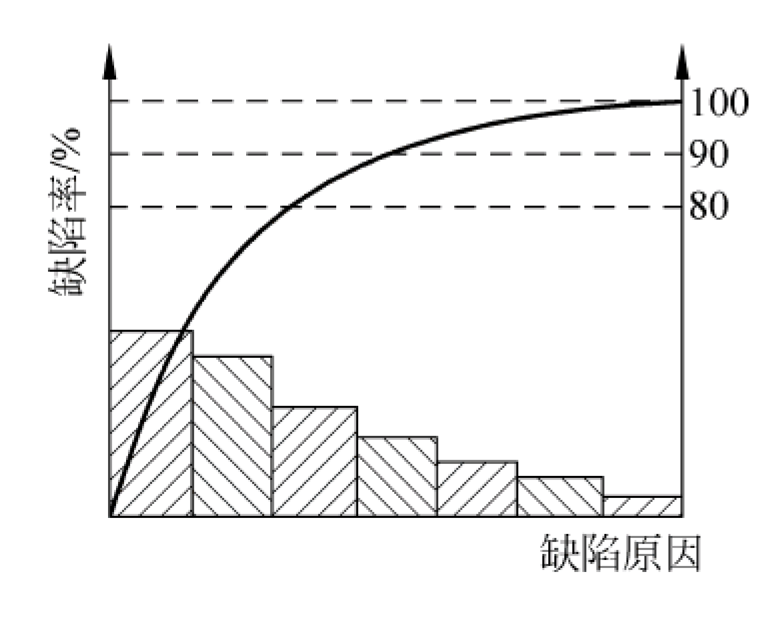


图 11.8 Pareto 图基本格式

3. 控制图和七点运行法则

控制图是数据的图形表示,是画有控制界线的图表,表明一个过程随时间变化的结果,用来分析质量波动究竟是由于正常原因引起还是异常原因引起,从而判明生产过程是否处

于控制状态。控制图的主要用途是预防缺陷,而不是检测或拒绝缺陷,可以帮助人们判断一个过程是在控制之中还是失去了控制。

质量控制图如图 11.9 所示。一般有三条线,上面一条虚线称为上控制界线(UCL),下面一条虚线称为下控制界线(LCL),中间一条实线称为中心线(CL)。将所控制的质量特性用圆点标记,若圆点全部在控制界线内,且排列无缺陷(如趋势、周期、接近),则可判断项目质量处于受控状态,否则认为项目实施存在异常,必须认真检查并予以消除。

人们在长期的实践中总结出七点运行法则,如果有连续的七个或七个以上的圆点分布在中心线的同一侧,或者出现同向变化的趋势,即使处于控制界线内,也表明出现了问题或受到了外界干扰,应视为失控状态。

4. 运行图

在软件项目管理中经常看到运行图的实例,运行图被用来把预测数据或历史记录数据进行比较,从而在某些方面解释所发生的情况。例如,通过运行图监视每星期出现的缺陷和在正式程序测试期间积累的缺陷,可作为实时的质量报告和工作记录。另外,可以追踪超过修正响应时间标准的软件修补百分比,从而保证及时地把修补发送到用户手中。逾期修正百分比运行图如图 11.10 所示。

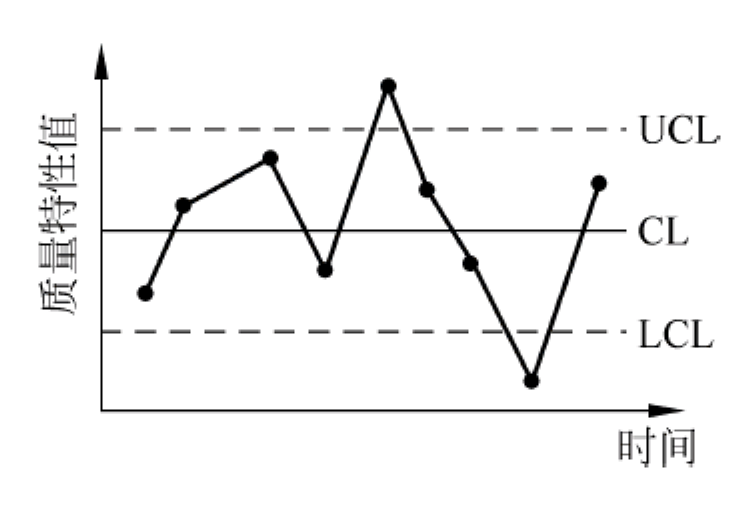


图 11.9 质量控制图

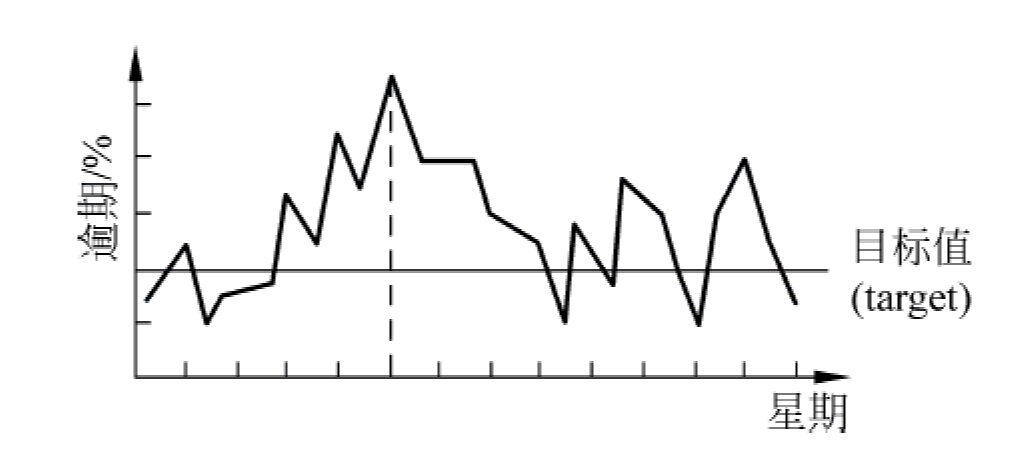


图 11.10 逾期修正百分比运行图

11.6 软件质量改进

为确保软件的有效性、效率和可追溯性,软件开发机构应识别关键质量要求,改进软件质量,以加强组织体系和提高软件满足用户要求的能力。

1. IDEAL 质量改进模型

SEI(美国卡耐基梅隆大学软件工程研究院)针对组织过程改进提出了 IDEAL 模型。IDEAL 模型将软件改进的整个过程分为五个阶段,是一个组织用于启动、规划和实现过程改善措施流图的模型,概括了建立一个成功的质量改进必要的步骤。其优点是简单易懂,方便实用。IDEAL 模型结构如图 11.11 所示。

各阶段的工作描述如下:

(1) 启动(Initiating,I)。为成功地改进软件质量工作奠定基础。

(2) 诊断(Diagnosing,D)。确定现状与质量目标之间的差距。

(3) 建立(Establishing,E)。建立如何达到质量目标的相关计划及方案。

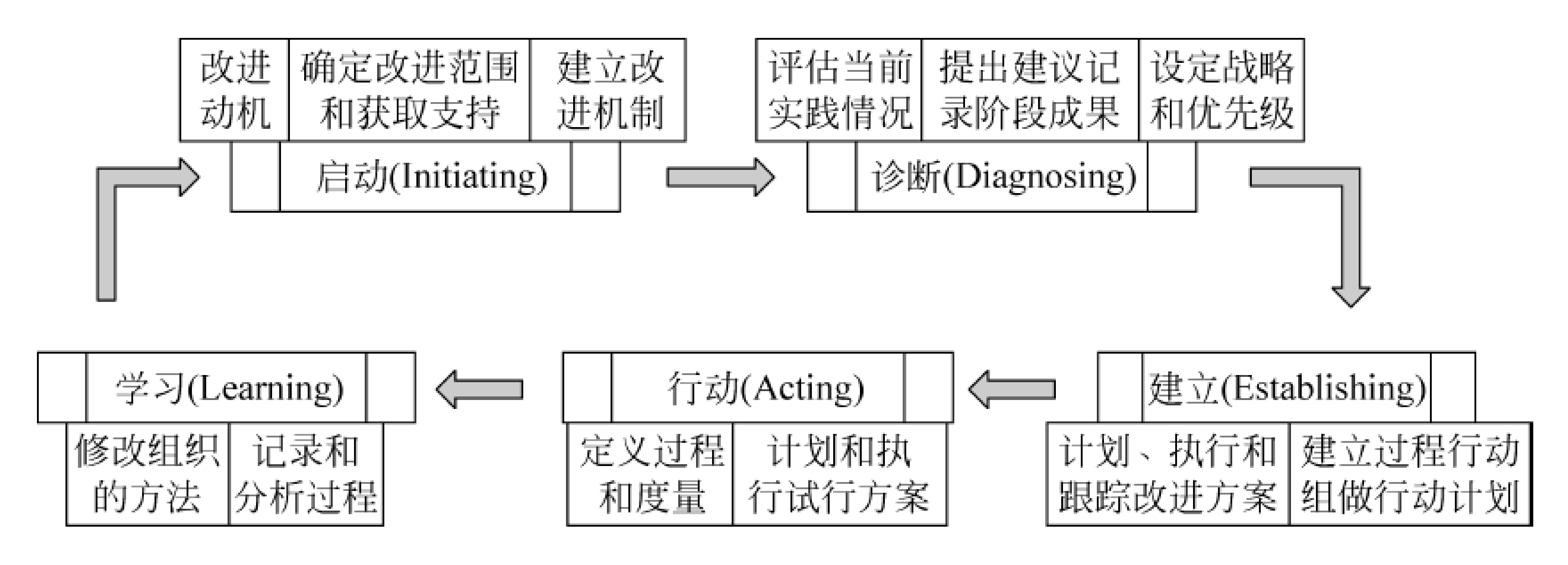


图 11.11 IDEAL 模型结构

(4) 行动(Acting,A)。根据计划及方案开展质量改进工作。

(5) 学习(Learning,L)。从经验中学习,以提高在将来采用新技术的能力。

2. 质量改进原则

根据软件质量改进工作的经验和教训,将软件质量改进归纳为"六要六不要"原则,具体描述如下:

(1) 要重视效果,不要徒有虚名。进行软件质量改进务必"头脑清醒",以改进效果好坏作为评价准则。那些徒有虚名而没有实际功底的企业,一定不会得到用户的长期"青睐",而失去客户则意味着失去一切。

(2) 要循序渐进,不要急于求成。软件质量改进是一个长期过程,不可能一朝一夕完成,要遵循质量改进的客观规律;违背客观规律而一味追求高级别的认证,对质量改进急于求成,往往欲速则不达。

(3) 要注重现实,不要"拿来主义"。由于企业的实际情况和文化背景不同,其他企业的成功经验在本企业往往不会"奏效",需要花费必要的时间和精力认真分析本企业的实际情况,建立适合本企业的管理制度和流程。照搬别人的做法只会导致失败。

(4) 要把握重点,不要遍地开花。大多数软件企业都是在生存中求发展,很难在人力、物力和财力上有足够的投入来进行全面的质量改进。需要结合本企业的实际情况,确定每个阶段质量改进的重点并"各个击破",可以在较短时间内收到明显效果。

(5) 要注重过程,不要只重结果。科学、严格的质量改进过程执行是保证结果的必要手段。质量过程改进的目的是为了取得良好结果,如果一味地追求结果而忽视对过程的改进和控制,必然收效甚微。只重结果是"短视"的,唯有注重过程才可能出现好结果。

(6) 要自我修炼和用户引导并举,不要一味"埋头苦干"。用户支持是质量改进获得成功的基础。企业自身"埋头苦干"的同时,一定要引导用户配合质量改进工作,让用户明确质量改进的目的和延长交付时间的原因,在工作上争取主动。

11.7 软件评审

软件评审是软件质量保证的重要措施。狭义的"软件评审"通常指软件文档和源程序评审;广义的"软件评审"还包括与软件测试相结合的评审以及管理评审。

11.7.1 评审内容

软件评审的内容很多，大体可分为管理评审、技术评审、文档评审和过程评审。

1. 管理评审

管理评审就是高层管理者针对质量方针和目标，对质量体系的现状和适应性进行正式评价。管理评审是以实施质量方针和目标等质量体系的适应性和有效性为评价基准，对体系文件的适应性和质量活动的有效性进行评价。体系审核的结果有时是管理评审的输入，即管理评审要对体系审核的“过程”和“结果”进行检查和评价。管理评审流程如图 11.12 所示。

2. 技术评审

技术评审是对软件及各阶段的输出内容进行评估，确保需求说明书、设计说明书与要求保持一致，并按计划对软件实施了开发。技术评审分为正式和非正式两种，通常由技术负责人制定详细的评审计划，包括评审时间、地点以及所需的输入文件。

技术评审结束后，以书面形式对评审进行总结，形成技术评审报告。技术评审报告的主要内容包括会议基本信息、存在的问题和建议措施、评审结论和意见、问题跟踪表、技术评审答辩记录等。

3. 文档评审

文档评审分为格式评审和内容评审。格式评审是检查文档格式是否满足要求；内容评审主要从以下方面进行检查：正确性、完整性、一致性、有效性、易测性、模块化、清晰性、可行性、可靠性、可追溯性等。

4. 过程评审

过程评审是对软件开发过程进行评审，通过对流程监控，保证软件质量组织制定的软件过程在软件开发中得到遵循，同时保证质量方针得到更好的执行。评审对象是质量保证流程，而不是产品质量或其他形式的产品。过程评审流程如图 11.13 所示。过程评审作用如下：

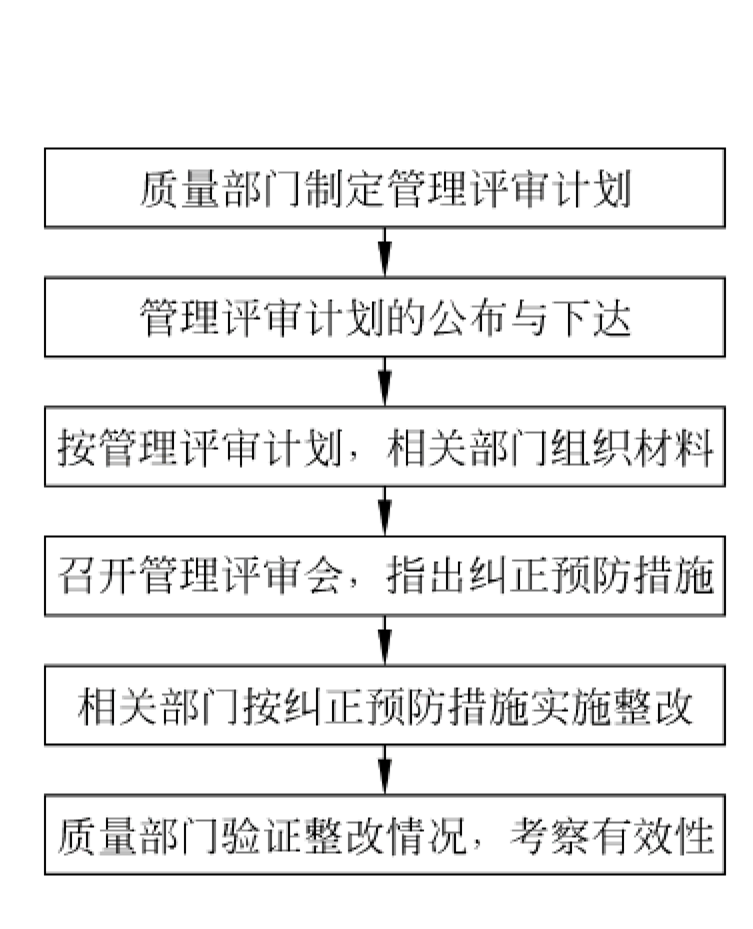


图 11.12 管理评审流程

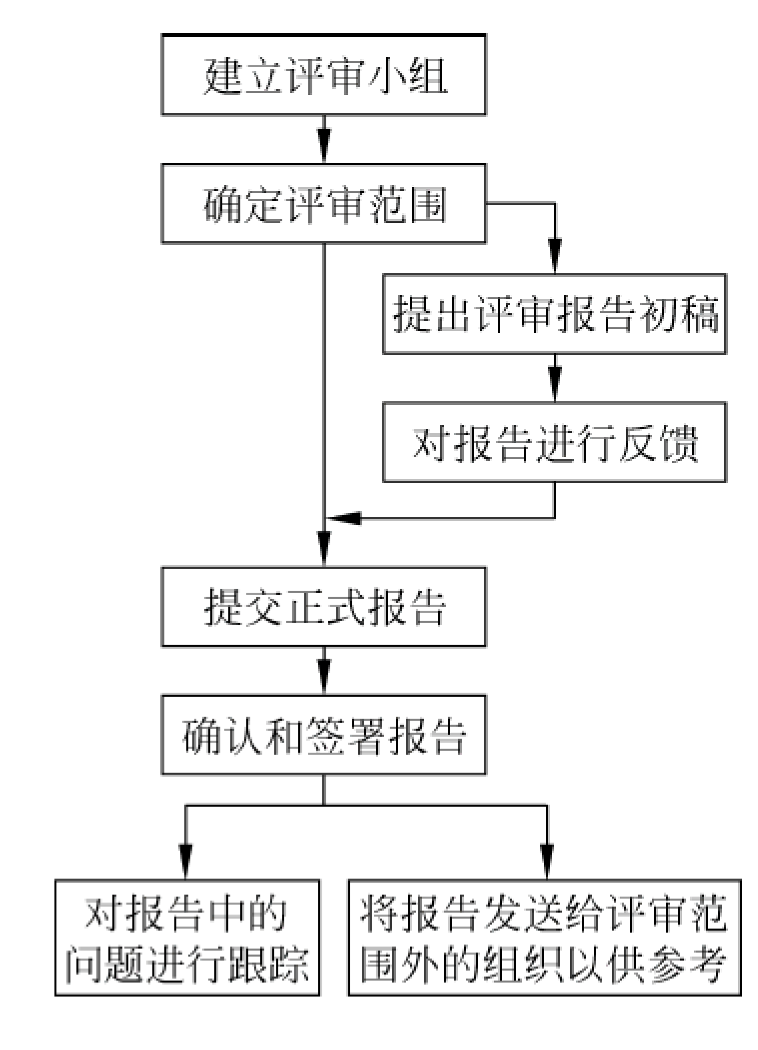


图 11.13 过程评审流程

(1) 评估主要的质量保证流程。

(2) 提出评审过程中出现问题的解决方法。

(3) 总结好的软件开发经验。

(4) 指出需要进一步完善和改进的部分。

11.7.2 评审方法

评审方法很多,分为正式评审和非正式评审。从非正式评审到正式评审如图 11.14 所示。

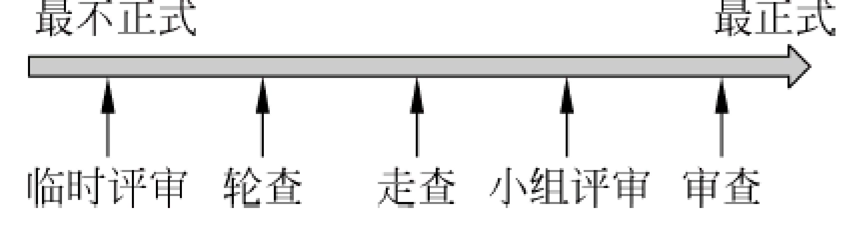


图 11.14 评审方法

1. 评审方法简介

(1) 临时评审。这是最不正式的评审方式,通常适用于小组间的合作。

(2) 轮查。将要评审的内容发送给各位评审员,并收集相关的反馈意见。

(3) 走查。这是常用的非正式评审方式,评审在作者的主导下进行。作者向评审员详细介绍软件产品,走查员针对发现的问题与作者沟通。由于在走查前没有要求走查员阅读软件产品,所以走查可能不够深入,一些隐藏较深的缺陷不易发现。

(4) 小组评审。这是比较理想的正式评审方式。相对于走查而言,主要改进是不再由作者主导评审过程,在评审会议前评审员需要对软件产品进行预评。这种方式既能提高评审质量,又能提高评审会议效率。但由于评审过程还不够完善,评审后期的问题跟踪和分析往往被简化和忽略。

(5) 审查。这是非常正式的评审方式,是最系统化、最严密的评审方法,评审效果最好。但持续时间较长,成本开销也较大,不一定是最经济的评审方式。审查的作用主要是:验证产品是否满足功能规格说明、质量特性以及用户需求;验证产品是否符合相关标准、规则、计划和过程;提供缺陷和审查工作的度量,改进审查过程和组织的软件工程过程。

2. 走查、小组评审和审查的比较

走查、小组评审和审查的比较如表 11.4 所示。

表 11.4 走查、小组评审和审查比较

序号	角色/职责	走查	小组评审	审查
1	主持者	作者	评审组长或作者	评审组长
2	材料陈述者	作者	评审组长	评审者
3	记录员	可能	是	是
4	专门的评审角色	否	是	是
5	检查表	否	是	是
6	问题跟踪和分析	否	可能	是
7	产品评估	否	是	是
8	计划	有	有	有
9	准备	无	有	有
10	会议	有	有	有
11	修正	有	有	有
12	确认	无	有	有

11.8 ISO 9000质量管理体系

质量管理体系认证作为合格评定的重要内容,已经得到国际社会的认可并迅速发展。有效的质量管理体系认证是提高市场竞争力的有效手段之一,是中国软件企业获得生存和发展空间的前提,是进入国际化大市场的必由之路,对促进软件外包和国际化软件开发具有重要意义。建立并实施质量管理体系,软件企业对内实施质量管理、对外证实质量保证能力,是在软件企业内部实施规范化管理的基础。

国际标准化组织(International Organization for Standardization,ISO)是国际标准化领域最重要的组织。ISO的任务是促进全球范围内的标准化及其有关活动,以利于国际间产品与服务的交流,以及在知识、科学、技术和经济活动中发展国际间的相互合作。ISO显示了强大的生命力,吸引了越来越多的国家参与。

11.8.1 ISO 9000族标准的组成

ISO 9000族标准由五部分组成:质量术语标准、质量保证标准、质量管理标准、质量管理标准和质量保证标准选用与实施指南、支持性技术标准。其组成结构如图11.15所示。

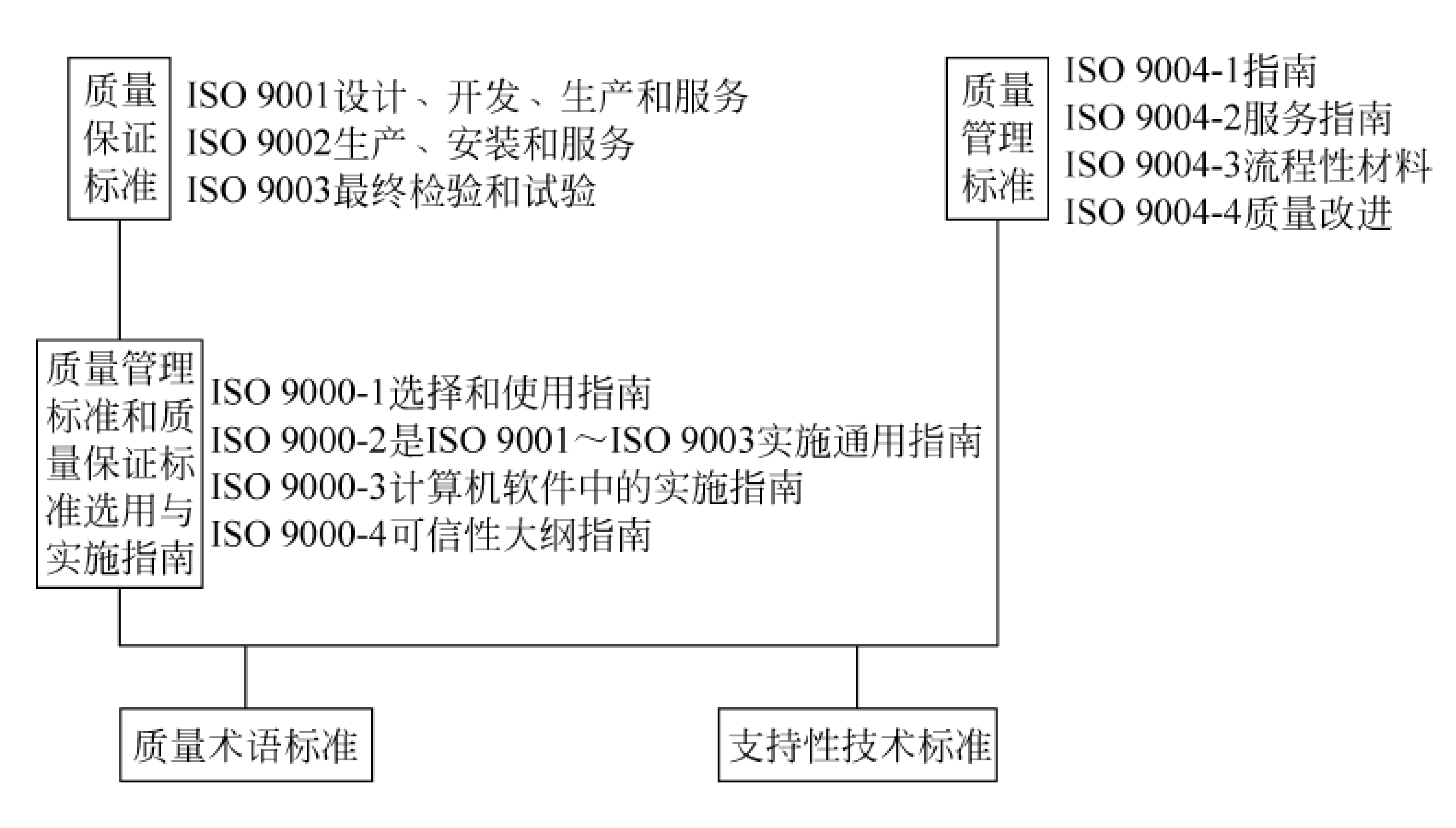


图11.15 ISO 9000族标准组成结构

企业活动一般由三方面组成:经营、管理和开发。在管理上又主要表现为行政管理、财务管理、质量管理等很多方面。ISO 9000族标准主要针对质量管理,同时涵盖了部分行政管理和财务管理的范畴。

具体来说,ISO 9000族标准在以下四个方面规范质量管理:

(1) 机构。明确规定为保证产品质量而必须建立的管理机构及其职责权限。

(2) 程序。企业组织产品生产必须制定的规章制度、技术标准、质量手册、质量体系操作检查程序等。

(3) 过程。质量控制是对生产全部过程进行控制,是面的控制,不是点的控制。并要求过程具有标识性、监督性、可追溯性。

(4) 总结。不断地总结、评价质量体系,不断地改进质量体系,使质量管理呈螺旋式上升。

11.8.2 ISO 9000 在软件组织的实施

随着软件行业的迅猛发展,软件组织的经营和运作模式发生了很大变化。许多软件组织已不仅仅局限于提供软件开发,还提供软件产品方案、软件开发咨询、代码编写等。不论软件组织以什么样的方式运作,所建立的质量管理体系必须是基于业务实现过程的,是为了保证组织既定的质量方针和目标的实现。

中国科学院软件研究所根据多年的软件开发和管理经验,提出了基于 ISO 9000 的软件质量保证框架,为我国中小规模的软件组织贯彻和实施 ISO 9000 标准提供了实施模型。该模型主要从软件过程管理的角度出发,指导软件组织导入 ISO 9000,建立符合 ISO 9000 的软件质量保证体系,并提供标准化的软件开发过程模板和自动化的软件管理过程模板,有效地进行过程控制,从而辅助组织进行过程改进和通过 ISO 9000 认证。该模型提出了符合 ISO 9000 的软件质量管理模式,如图 11.16 所示。

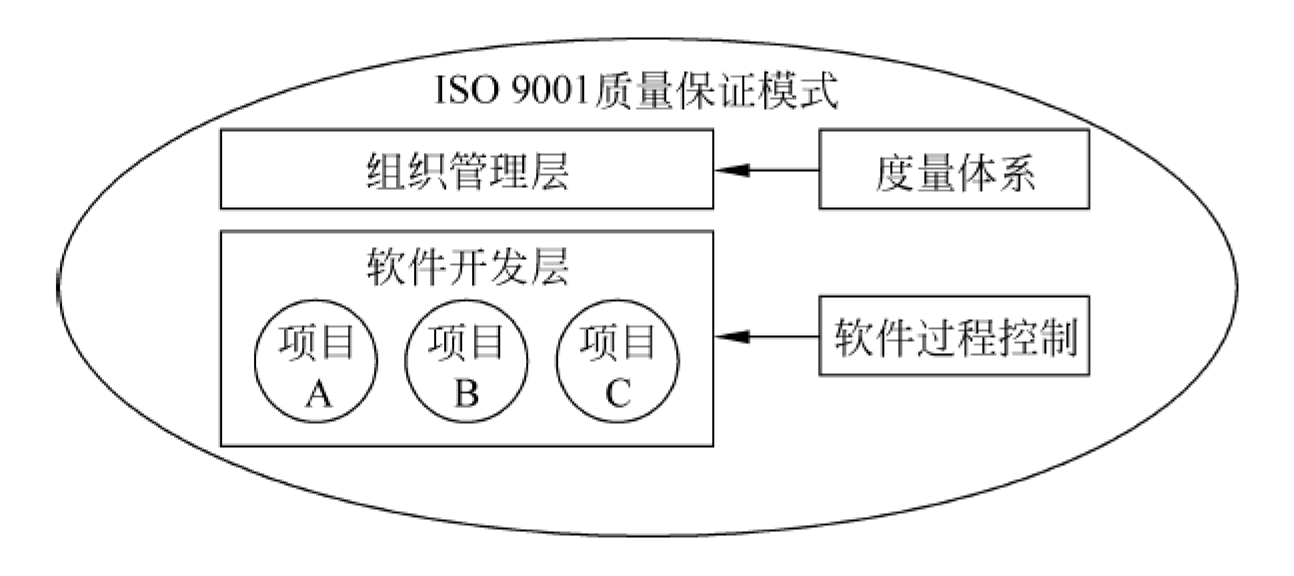


图 11.16 符合 ISO 9000 的软件质量管理模式

软件组织如何建立符合自身情况、基于 ISO 9000 的质量保证体系呢?首先必须建立有效的质量体系,并以文件化的形式形成制度。质量体系包括如下内容:

(1) 分配组织管理职责。对影响质量管理、实现和验证的相关人员,需要明确分配职责、职权和相互关系。

(2) 识别组织标准过程,确定软件生命周期模型。识别和确定过程是实现过程控制的基础,不能识别过程就无法定义过程,更谈不上对过程的控制;确定软件生命周期模型包含整个生命周期与软件产品开发、运行和维护有关的过程、活动和任务。

(3) 建立和发布质量体系文件,包括质量手册、质量体系程序文件和其他质量文件。

思考题

1. 如何理解软件质量的定义?
2. 理解 McCall 软件质量模型及质量特性属性。
3. 简述软件质量管理的主要内容。
4. 如何准确掌握项目质量策划的定义?

5. 质量策划的主要工作有哪些？
6. 简述质量计划的主要内容。
7. 如何理解软件质量保证？
8. 软件质量保证有哪些主要活动？
9. 软件质量保证有哪些关键技术？
10. 如何理解 IDEAL 质量改进模型？该模型包括哪些阶段？
11. 软件质量改进的“六要六不要”原则具体是什么？
12. 简述软件评审的内容。
13. 软件评审方法有哪些？如何选择评审方法？
14. 质量管理体系认证有什么作用？
15. ISO 9000 族标准在哪些方面规范质量管理？

第12章 成本管理

软件成本管理是指为了保障软件项目实际发生的成本不超过项目预算，在批准的预算内按时、按质、经济高效地完成既定目标而开展的成本管理活动。成本管理活动主要依靠资源计划、成本估算、成本预算、成本控制四个过程来完成。资源计划是指完成项目需要资源的种类、数量和时间；成本估算是估计总成本和误差范围；成本预算是将总成本分配到各项工作任务中去；成本控制是控制成本的偏差，分析原因和采取措施以确保将总成本控制在预算以内。

12.1 软件成本分析

12.1.1 软件成本特点

软件成本是在整个生命周期内发生的全部支出之和，不同于其他产品生产经营过程中发生的各项费用支出，特点如下：

（1）费用支出具有较高的风险性。软件开发费用支出一般比较大，开发周期长，因此，对费用支出事先要进行合理研究，使风险尽可能降低。

（2）费用支出和经济效益具有不确定性。这种不确定性主要体现在两个方面：一是系统开发存在成功与失败的可能性，费用支出能否取得预期成果事先无法确定；二是即使系统开发成功，未来创造多少经济效益事先也难以确定。

（3）费用支出具有资本性支出的性质。软件成本费用在开发期和使用期都要发生，主要发生在开发期，这些成本费用虽在本期发生，但研究成果一般主要在以后各期发挥作用，与以后各期的收益相关。即使开发失败，费用支出是为了以后各期，同样具有资本性支出的性质。

（4）费用支出将形成企业资产。软件成本费用发生的目的是使预期成果在企业生产经营过程中得以运用，并为企业带来经济效益。如果研究成功，则软件成为企业资产；即使失败，某些支出也会形成企业资产，如硬件成本和准备费用等。

12.1.2 软件成本构成

对于一般项目，按照成本与产品的关系，可将成本划分为直接成本和间接成本。直接成本也称为直接费用，间接成本称为间接费用。直接成本与间接成本各有两种含义：从成本与开发过程的关系来看，是指直接生产成本与间接生产成本；从费用计入成本的方式来看，

是指直接计入成本与间接计入成本。

结合软件项目特点，软件成本主要由以下几部分构成：

（1）劳动力成本。软件研制主要依靠人的脑力劳动，其中绝大部分工作都是靠人来完成，劳动力成本占软件成本的绝大部分。软件开发人员一般是高学历、高层次的人才，这些人对薪金要求比较高，当他们在企业连续工作两三年，技术和业务水平得到提高并趋于成熟后，就会产生加薪的愿望，一旦企业不能满足，有些人就会跳槽，而企业为防止人才流失，只能以支付高薪为代价。另外，软件企业为了追求软件开发技术的先进性和满足员工在职提高的愿望，常常为员工提供各种培训契机，高额的培训费、差旅费、补助费等一般由公司承担，而技术为员工所有，一旦公司在工资、福利、职务晋升等方面不能满足其要求时，就会出现人员流失现象，致使公司所付出的高额培训成本不能完全得到回报。

（2）设计开发成本。设计开发成本是指软件开发过程中发生的论证费、调研费、计算费、技术资料购买费、复制和翻译破译费、考察费、授课费、咨询费、设计用品费、设计评审费、用户培训费等。软件按事先是否针对具体用户可分为两类：一类是针对具体用户需求而设计开发的产品，按照软件合同进行系统分析、设计、实施、测试、维护，当软件开发进度不能按照预定期限完成时，不仅增加了开发过程的劳动力成本、管理费用等额外支出，而且还影响后续项目的开展及其他项目的进程；另一类是软件企业为适应市场而开发的新产品，其突出特点是不确定因素较多，风险较大，投入的开发成本、营销成本、管理成本等能否收回，完全依赖市场，不能收回的部分将成为永久的沉淀成本。

（3）硬件成本。硬件成本主要包括开发软件所需要的硬件设备、系统软件、数据资源购置、运输、仓储、安装、测试等费用。软件开发所需要的物质类资产大部分为电子类产品，电子类产品的性能特点决定其使用寿命短、更新快，软件企业在生产经营过程中要不断地消耗和更新硬件材料，才能维持正常的经营运作。若内部管理不完善，从采购到保管、领用、监控等环节有纰漏，则会使这部分成本增加。

（4）税金成本。软件税金主要有三项：增值税、营业税、企业所得税。国家为了鼓励软件企业发展，颁布了一些税收优惠政策，例如，一般增值税纳税人实际税负超过3%的部分实行即征即退的优惠政策；新创办软件企业经认定后，自获利年度起，享受企业所得税“两免三减半”的优惠政策；软件企业人员薪酬和培训费用可按实际发生额在企业所得税税前列支；对重点软件企业所得税、软件企业进口自用设备关税和增值税也都有相应的优惠政策。充分运用这些优惠政策，可减少软件税金成本，为软件企业发展壮大提供有利的宏观环境。

（5）管理费用、财务费用成本。管理费用包括水电费、房租费、通信费、办公费、职工福利费、广告费及管理人员工资等。尽管与其他类型企业相比，软件企业管理费用相对较低，但在电费、通信费、管理人员工资等方面通过有效的管理与控制，还可进一步压缩成本。对财务费用，其中筹资、融资费用具有一定弹性空间，银行手续费等方面一般情况下变化不大。

（6）服务成本。软件产品交付用户后，一般提供一年左右的免费维护期。由于软件产品的特殊性（一般为应用软件），与其他产品相比，在服务内容、服务方式、服务成本等方面具有较大差异。在软件免费维护期间，软件企业不仅要承担软件维护的劳动力成本、差旅费成本、硬件更换成本，而且要长期不间断地跟踪服务，这样加大了维护成本，特别是在软件开发人员离职，其他技术人员不熟悉该软件开发细节的情况下，给软件维护工作带来一定困难，致使软件企业承担更大的服务成本，软件成本也相应增加。

12.1.3 软件成本影响因素

软件成本的影响因素很多,主要包括以下几个方面:

(1) 质量对成本的影响。质量对成本的影响可以通过质量与成本的关系表示,如图 12.1 所示。

质量总成本由质量故障成本和质量保证成本组成。质量故障成本是指为了排除产品质量原因所产生的故障,保证产品重新恢复功能的费用;质量保证成本是指为了保证和提高产品质量所采取的技术措施而消耗的费用。

质量保证成本和质量故障成本是相互矛盾的。质量越低,由于质量不合格引起的损失就越大,故障成本增加;质量越高,相应的质量保证成本也越高,故障就越少,由故障引起的损失也相应减少。因此,需要建立一个动态的平衡关系。

(2) 工期对成本的影响。项目成本由直接成本和间接成本组成,工期越长,直接成本越低,间接成本越高;工期越短,直接成本越高,间接成本越低。工期与成本相互间的关系如图 12.2 所示。对于软件项目,工期长短对成本的影响很大,缩短工期,需要更多的、技术水平更高的工程师,直接成本就会增加。

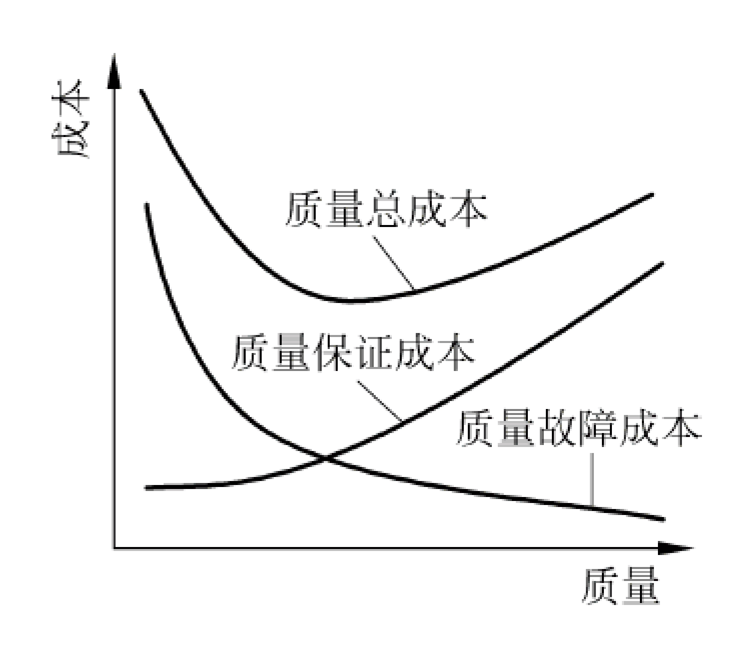


图 12.1 质量与成本的关系

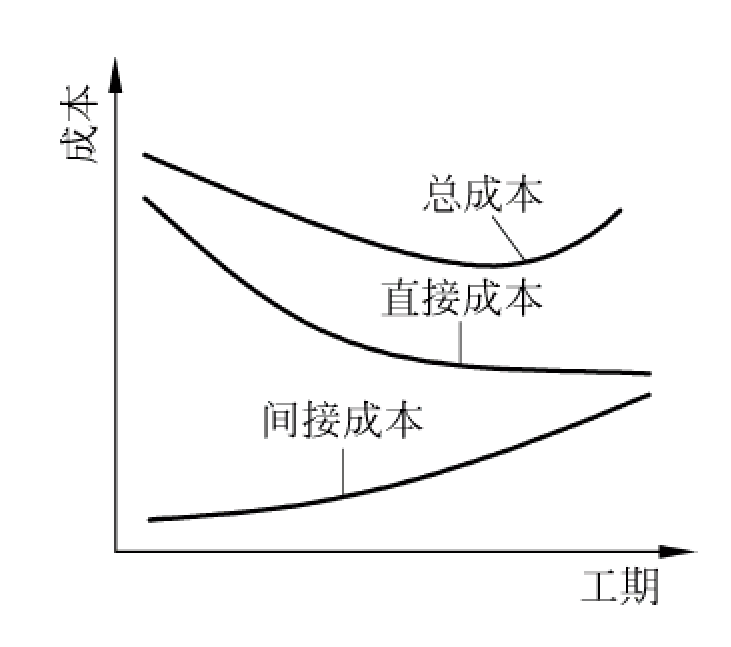


图 12.2 工期与成本的关系

(3) 管理水平对成本的影响。软件开发与实施的管理水平对成本也产生影响,有时还是根本性的。较高的管理水平可以提高预算的准确度,加强对预算的执行和监督,对工期的控制严格限制在计划许可范围内,对设计方案和项目计划更改造成的成本增加、减少和工期变更,可以较为有效地控制。而对风险的识别、采取的措施,高水平的项目管理也会达到减少风险损失的效果。

软件项目计划变更比较多,有项目组的因素,更多的是来自用户方面。项目经理要求对计划变更有严格的评审和审查程序。对项目经理而言,仅单纯地限制开发成本并不能从根本上解决问题,应提高计划制定的准确性和执行的严肃性。

(4) 人力资源对成本的影响。人力资源对成本的影响表现在两个方面。一是人力资源素质。对高技术能力、高素质的人才,本身的人力资源成本比较高,但可以产生高的工作效率、高质量的产品、较短的工期等间接效果,从而总体上降低成本;而对于一般人员,需要技术培训,对软件的理解及工作效率相对低下,导致工期延长,需要雇佣更多人员,造成成本增加。二是人员的不确定性。软件项目最主要的生产力是人,人具有易变的属性,不像操纵机器设备那样简单。人的积极性、人员流动、团队氛围等对成本有重要影响。

(5) 需求不确定性对成本的影响。在软件开发过程中,用户需求往往是经常变动的,主要是由于用户和开发人员交流不够充分,用户认识问题过程的循序渐进所造成的。需求的不确定在成本管理方面表现为项目时间的延长和人员的增加,从而导致成本增加。

(6) 价格对成本的影响。中间产品和服务、人力资源以及硬件、软件的价格也对成本产生直接影响。价格对软件成本估算影响很大。

12.2 软件资源计划

完成软件项目必须消耗劳动力(人力资源)、材料、设备、资金等有形资源,同时还可能消耗无形资源。由于存在资源约束,耗用资源的质量、数量、均衡情况等对工期、成本有着不可估量的影响。在任何项目中,资源并不是无限制的,也并不是可以随时随地获取的。在软件项目管理过程中,资源能够满足需求的程度以及资源与进度的匹配是成本管理必须计划和安排的工作。如果资源配置不合理或使用不当,会使工期拖延或成本超支。

1. 编制步骤

资源计划编制通常分为以下四个步骤:

(1) 资源需求分析。通过分析确定工作分解结构中每一项任务所需的资源数量、质量及其种类,根据项目的资源消耗定额或经验数据,确定资源需求量。一般按以下步骤进行:

① 工作量计算。

② 确定实施方案。

③ 估计人员需求量。

④ 估计设备、材料需求量。

⑤ 确定资源的使用时间。

(2) 资源供给分析。资源供给的形式多种多样,可以在组织内部解决,也可以从组织外部获得。资源供给分析要分析资源的可获得性、获得的难易程度以及获得的渠道和方式。

(3) 资源成本比较与资源组合。比较资源的使用成本,确定资源的组合模式(即各种资源所占比例与组合方式)。完成同样的工作,不同的资源组合模式,成本可能会有很大的差异。要根据实际情况,考虑成本、进度等目标,来确定资源的组合模式。

(4) 资源分配与计划编制。资源分配是一个系统工程,既要保证各个任务得到合适的资源,又要努力实现资源总量最少、使用平衡。在保证所有任务都分配到所需资源、所有资源都得到充分利用的基础上,编制资源计划。

2. 编制方法

资源计划编制方法很多,最常用的是德尔菲评估法,也称专家评估法,是指由成本管理专家根据经验和判断来编制项目资源计划的方法。这种方法通常又分为两种具体形式:

(1) 专家小组法。专家小组法是组织一组有关专家在调查研究的基础上,通过召开专家小组座谈会的方式共同探讨,提出资源计划方案,然后确定编制资源计划的方法。

(2) 德尔菲法。德尔菲法是采用函询调查的方法,将讨论的问题和必要的背景材料编制成调查表,采用通信的方式寄给各位专家,利用专家的智慧和经验进行信息交流。将专家

的意见进行归纳、整理后反馈给专家，再次征求意见，然后再进行归纳和反馈。这样经过多次循环后，就可以得到比较一致且可靠性较大的意见。

3. 编制工具

常用的资源计划编制工具包括资源矩阵、资源数据表、资源甘特图、资源负荷图、资源累计需求曲线等。其中，资源矩阵、资源数据表是以表格的形式显示项目的任务、进度及其需要资源的品种、数量等，格式如表 12.1、表 12.2 所示；资源甘特图是利用甘特图技术对资源的需求进行表达，格式如图 12.3 所示；资源负荷图是以条形图的方式反映项目进度及其资源需求情况，格式如图 12.4 所示；资源累计需求曲线是以线条的方式反映项目进度及资源累计的需求状况，格式如图 12.5 所示。

表 12.1 资源矩阵

任务	资源需求					相关说明
任务 1 任务 2 ⋮ 任务 n	资源 1	资源 2	资源 3	…	资源 n	

表 12.2 资源数据表

需求种类	需求总量	时间安排(不同时间段资源需求量)				相关说明
		1	2	…	T	
资源 1 资源 2 ⋮ 资源 n						

资源种类	时间安排(不同时间段资源需求量)											
	1	2	3	4	5	6	7	8	9	10	11	12
资源 1 资源 2 ⋮ 资源 n												

图 12.3 资源甘特图

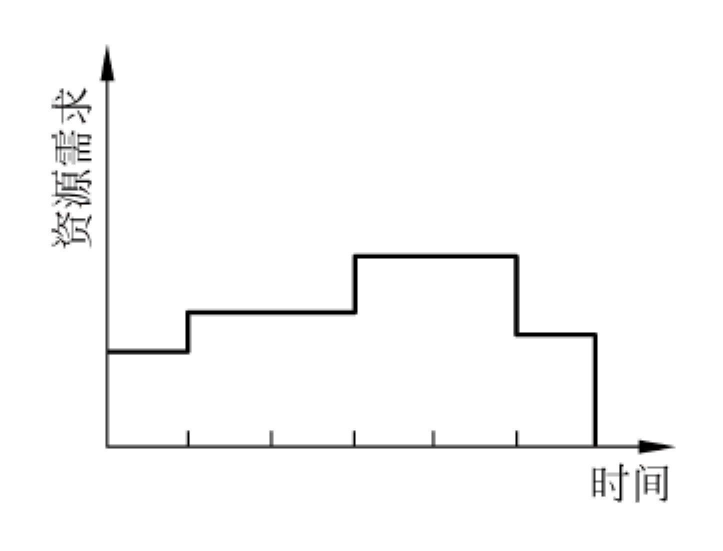


图 12.4 资源负荷图

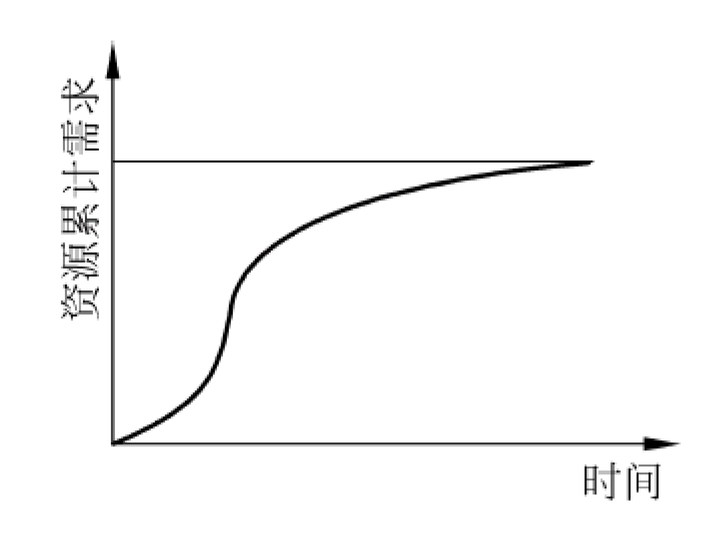


图 12.5 资源累计需求曲线

12.3 软件成本估算

准确的成本和进度估算是保证软件开发和应用顺利进行的必要手段,是软件前期的基础工作,是软件工程领域重要的研究方向。软件成本包括开发成本和维护成本,成本计量单位可用“人月”来表示,也可用开发与维护过程中投入的人力、物力、财力折合成的货币单位表示。软件产品主要消耗人的劳动,不像传统的工业产品那样主要消耗原材料和能源;软件产品是高智力的无形产品,不像传统产品那样有明显的大小。所以,软件产品成本估算比传统工业产品成本估算更为复杂。

12.3.1 代码行法软件规模估算

代码行(Line Of Code,LOC)法是传统的估算方法,尤其适合于过程化语言。有几种原始资料可以用于估算新代码行,其中最好的是历史数据。在项目早期可能会有功能点、组件或任何早期可利用的数据,都能转换成代码行数。在缺乏历史数据时,可用专家意见估算出可能的、最不可能的、最有可能的代码行数,通过计划评审技术(Program Evaluation and Review Technique,PERT)估算出代码行数。

源代码行一般不包括未交付的支持软件,如测试驱动程序。如果这些未交付支持软件的开发与交付软件的开发一样仔细,需要有自己的评审、测试、文档等,那么也应该计算在内,目的是度量投入程序开发中的智力工作量。

定义一行代码是有困难的,这涉及不同语言之间可执行语句和数据声明概念上的差异。为跨越不同编程语言,定义一致的度量标准,采用逻辑源语句,并根据计算源语句定义检查表进行判断,如表 12.3 所示。

表 12.3 计算源语句定义检查表

序号	检查类别	源语句检查规则(同一类别多个规则是“或”的关系)
1	语句类型	可执行的;不可执行的申明或编译指令
2	如何产生	编程;用自动转换器转换;复制或不做修改地利用;修改
3	起因	新任务;以前的版本、构造或发布;以前的复用库;以前的其他软件组件或类库
4	用法	在主要的产品中,或作为其中一部分
5	交付	交付为源代码
6	功能性	运行中;起作用(有意的死代码,在特殊情况下激活)
7	复制	原始代码;对存储在主代码中的原始语句进行物理复制
8	开发状态	已完成系统测试
9	语言	对每种语言分别合计
10	说明	Null、Continue 和空操作;使用 Begin…End 或{…}对,表示可执行语句;Elseif 语句;关键字,如过程划分、接口和实现

来源:对 *Software Engineering Economics* 中的表 2-53 进行整理得来。

统计代码行费时费力,自动统计工具很多,常用的有以下两个:

(1) 代码行统计小助手。这个工具用于对单个文件或整个目录文件的代码行进行统计,适用于 Java、C、C++、Delphi、VB、C#、PB、SQL 等开发工具编写的源程序文件。统计结

果包括总代码行数、程序代码行数、注释行数和空行数及其所占的百分比。可将统计结果保存为 PDF 或 TXT 文件，支持结果打印，适用于 Windows 系列操作系统和 Linux 操作系统。

（2）智能源码统计专家。这个工具可用于对 VC++、C++ Builder、Delphi、VB、C/C++、ASM、Java 等程序源码进行详细统计，可以非常准确地分析出程序中代码行、注释行和空白行的数量。程序会自动根据文件类型选择相应的统计方式，并将所有文件的分析结果进行汇总，便于直观地对程序代码量进行全面统计。该软件是绿色软件，不需要安装，展开到任意目录，直接运行即可。

代码行法简单易行，但是该方法与程序设计语言的功能和表达能力密切相关，在软件项目开发前或开发初期估算代码行十分困难，非常适用于过程化程序设计语言，对非过程化程序设计语言则有一定的局限性。

12.3.2　功能点分析法软件成本估算

功能点分析（Function Point Analysis，FPA）法是在软件开发过程中完成软件估计、度量、分析等项目管理活动，其目的在于帮助软件项目解决管理方面的问题，促进项目的规范化，提高用户满意度，创造新的市场机会。与传统的代码行估算方法相比，功能点分析法是从用户角度来估算软件规模，不依赖于编程语言，在早期就可以对软件规模进行估算，并在开发过程中不断地更新数据，从而实现持续一致的管理。目前，功能点分析法是最重要、最有效的软件规模估算方法之一，在软件规模估算中起着重要作用。

功能点分析法分为以下三个步骤。

第一步：未调整功能点数估算

未调整功能点数（Unadjusted Function Points，UFP）的估算通过以下几步进行：

（1）定义计算边界。标识要度量的应用程序与外部应用程序和用户区域的边界。对于计算机系统来说，同其他计算机系统交互非常普遍，必须从用户角度划分边界，确定计算范围，如图 12.6 所示 ，用户功能点类型如表 12.4 所示。

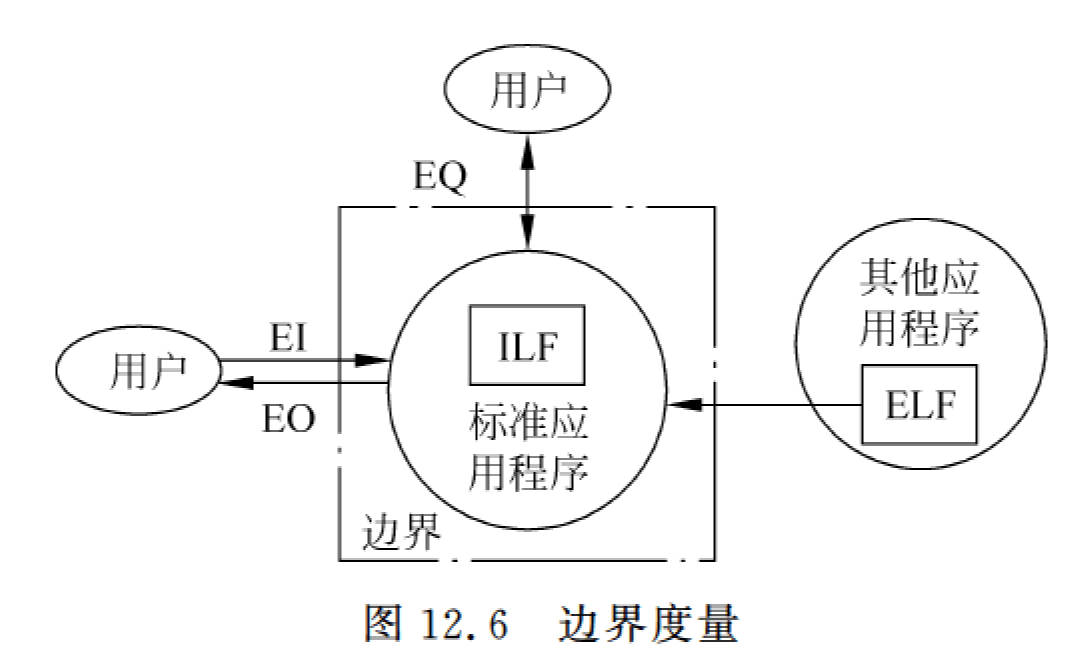


图 12.6　边界度量

表 12.4　用户功能点类型

序号	功　能　点	英文缩写	英 文 全 称
1	外部输入	EI	External Input
2	外部输出	EO	External Output
3	外部查询	EQ	External Query
4	内部逻辑文件	ILF	Internal Logical File
5	外部接口文件	EIF	External Interface File

(2) 按类型确定功能点数。根据软件需求和设计文档,明确划分表12.4中的五种用户功能点类型,对每种类型功能点数分别统计。

(3) PERT每类功能点数量估算。对每类功能点要产生三个估算量。

① 乐观功能点数 a_i:在最有利的情况下,第 i 类功能点的最低规模。

② 最可能功能点数 m_i:在正常情况下,第 i 类功能点的最可能规模。

③ 悲观功能点数 b_i:在最不利的情况下,第 i 类功能点的最高规模。

每类功能点数的期望值为 E_i:

$$E_i = \frac{a_i + 4m_i + b_i}{6} \tag{12.1}$$

(4) 确定复杂性等级。每个功能点赋予一个功能点复杂性等级。功能点复杂性等级由数据元素类型(Data Element Type,DET)、记录元素类型(Record Element Type,RET)和参考文件类型(Reference File Type,RFT)的数目决定。

① 数据元素类型(DET):一个DET就是一个唯一的用户可辨认的、不可递归的域。例如,一个通信地址由省、市、县、镇、村、组、邮政编码七个域组成,可以计算为七个DET。

② 记录元素类型(RET):一个RET就是一个用户可辨认的ILF或EIF中的数据元素组成的子组。例如,人员信息(雇员、顾客)可以计算为两个RET。

③ 参考文件类型(RFT):指可维护、读取、参考的ILF和可读、参考的EIF的数目。例如,一个EQ基本处理需要来自两个EIF的数据,可作为两个RFT计算。

每个功能的复杂性等级具体确定为"低"、"一般"、"高"三个等级。各种类型功能点的复杂性等级如表12.5、表12.6和表12.7所示。

(5) 计算未调整功能点数。对于表12.5、表12.6和表12.7中的每个UFP复杂性等级有一个对应的复杂性权重,如表12.8所示。

表12.5 EI的复杂性等级

参考文件类型(RFT)	数据元素类型(DET)		
	1～4	5～15	>15
0～1	低	低	中
2	低	中	高
≥3	中	高	高

表12.6 EO和EQ的复杂性等级

参考文件类型(RFT)	数据元素类型(DET)		
	1～5	6～19	>19
0～1	低	低	中
2～3	低	中	高
>3	中	高	高

表 12.7　ILF 和 EIF 的复杂性等级

记录元素类型（RFT）	数据元素类型(DET)		
	1～19	20～50	＞50
0～1	低	低	中
2～5	低	中	高
＞5	中	高	高

表 12.8　复杂性权重

功能点类型	复杂性权重		
	低	中	高
外部输入	3	4	6
外部输出	4	5	7
外部查询	3	4	6
内部逻辑文件	7	10	15
外部接口文件	5	7	10

未调整功能点数是通过表 12.9 计算出来的。未调整功能点数栏的计算公式为低、中、高功能点的数量分别乘以其权重之和。即：

$$未调整功能点数 = 数量(低)\times 权重(低) + 数量(中)\times 权重(中) + 数量(高)\times 权重(高) \quad (12.2)$$

表 12.9　未调整功能点数估算表

功能点类别	数量	低		中		高		未调整功能点数(UFP)
		数量	权重	数量	权重	数量	权重	
外部输入			3		4		6	
外部输出			4		5		7	
外部查询			3		4		6	
内部逻辑文件			7		10		15	
外部接口文件			5		7		10	
总计数值			****		****		****	

第二步：调整后功能点数估算

UFP 是通过建立一个标准来确定某个特定的测量参数进行计算，复杂性权重的确定带有一定的主观性。UFP 与功能点调整系数(Function Point Adjust Factor，FPAF)相乘得到调整后的功能点数作为软件规模估算的功能点数。

FPAF 通过技术复杂因子(Technical Complexity Factor，TCF)进行计算。

TCF 共由三大类 14 个因子组成，如表 12.10 所示。每个因子按照其对系统的重要程度分为六个级别，如表 12.11 所示。每个因子的记分指南见参考文献《功解点分析方法与实践》。

表 12.10 技术复杂因子组成

因子大类	系统复杂度	输入和输出复杂度	应用软件复杂度
因子名称	1. 数据通信 2. 分布式处理 3. 性能 4. 配置项负载	5. 事务率 6. 在线数据项 7. 用户使用效率 8. 在线更新	9. 复杂处理 10. 重用性 11. 安装难易程度 12. 操作难易程度 13. 多个地点 14. 修改难易程度

表 12.11 权重表(F_i 的取值)

0	1	2	3	4	5
没有影响	偶有影响	轻微影响	一般影响	较大影响	严重影响

FPAF 用下式计算：

$$FPAF = 0.65 + 0.01 \times \left(\sum_{i=1}^{14} F_i \right) \tag{12.3}$$

调整后的功能点数 FP 用下式计算：

$$FP = UFP \times FPAF \tag{12.4}$$

第三步：成本估算

软件开发包括需求、设计、编码、测试、评审以及项目管理等所需要的时间。软件生产率的影响因素很多，每个软件组织需要根据自身的具体情况进行分析，这需要大量的历史数据作基础，对于缺乏类似数据的组织来说，找出生产率因素并不容易。根据相关参考文献及作者本人的开发经验，给出了当前常用开发工具中一般水平开发人员的生产效率(每人日完成的功能点数)，如图 12.7 所示。根据软件的功能点数和生产率，可估算出软件的开发周期和成本。

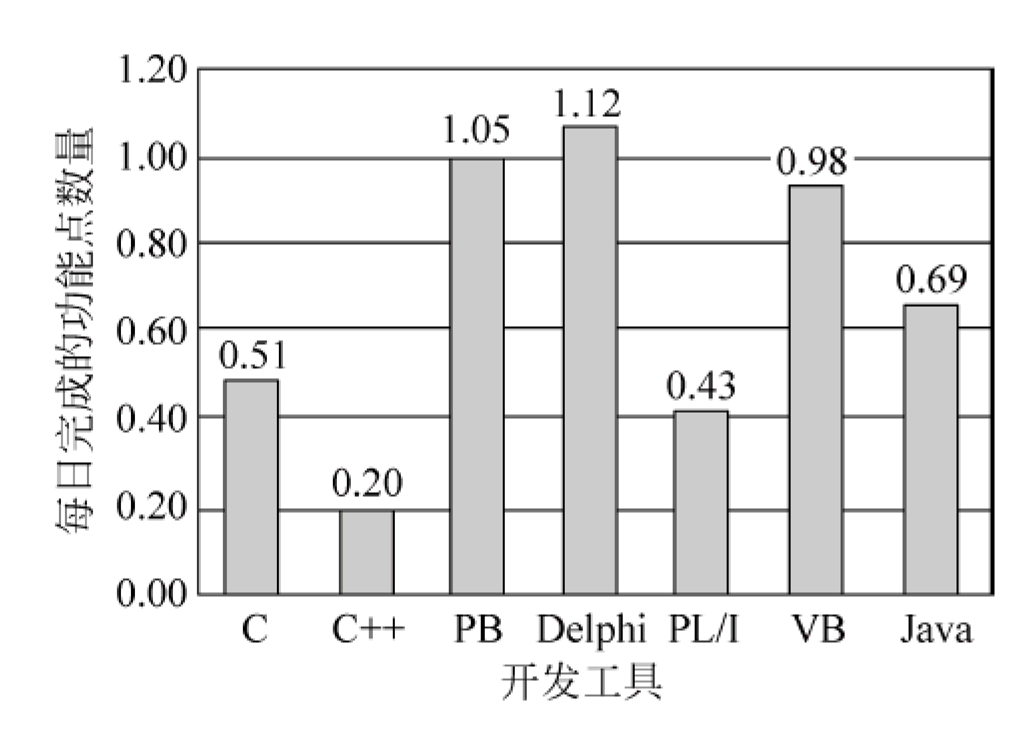


图 12.7 各种开发工具的日生产率

估算软件成本的计算公式为：

$$\text{软件开发成本(PM)} = \text{FP}/(\text{开发工具的日生产率} \times 19) \tag{12.5}$$

式(12.5)中，一个月的实际工作天数按 19 天计算。

公式结果以人月(Person Month，PM)作为计量单位。如果改为货币单位，可用人月乘

以劳动力月成本。

假定一个软件项目 FP 的估算值为 1000，采用 Java 语言开发，则开发成本以人月为单位的估算值为 1000/(0.69×19)=76(PM)。如果劳动力月成本为 6000 元人民币，软件开发成本以货币为单位的估算值为 76×6000=456000(元)。

根据以人月(PM)为单位的软件开发成本，并结合开发中可供安排的人员情况，可估算出软件的开发周期并制定相应的进度计划。

12.4 软件成本预算

成本预算是在成本估算的基础上，更精确地估算项目总成本，并将其分摊到项目的各项具体活动和各个具体阶段上，为软件成本控制制定基准计划的软件成本管理活动。

成本估算的目的是估计软件的总成本和误差范围，而成本预算是将软件的总成本分配到各工作项和各阶段上。成本估算的输出结果是成本预算的基础和依据，成本预算则是将已批准的估算进行分摊。由于预算时不可能完全预计到实际工作中遇到的问题和所处的环境，所以实际成本与预算计划成本发生偏离在所难免。如果出现偏离，就需要对相应的偏离进行考察，以确定是否会突破预算约束，管理者可以更清楚地掌握项目进展和资源使用情况，避免造成项目失败或效益不佳等后果。

1. 成本预算的特征

成本预算具有以下特征：

(1) 计划性。在软件项目计划中，根据工作分解结构(Work Breakdown Structures，WBS)将软件项目分解成多个工作包，形成一种系统结构。成本预算就是将成本估算总费用尽量精确地分配到 WBS 的每一个组成部分，从而形成与 WBS 相同的系统结构。因此，预算是另一种形式的软件计划。

(2) 约束性。因为软件项目高级管理人员在制定预算时通常希望尽可能“正确”，既不过分慷慨，以避免浪费或管理松散，也不过分吝啬，以避免任务无法完成或质量低下。所以，成本预算也是一种资源分配计划，结果可能并不能满足相关人员的利益要求，而表现为一种约束，相关人员只能在这种约束条件下工作。

(3) 控制性。项目预算实质是一种控制机制。管理者的任务不仅是完成预定目标，而且还必须具有效率，尽可能在完成任务的前提下节省资源，获得最大的经济效益。管理者必须小心谨慎地控制资源，不断根据进度检查使用的资源量，如果出现与预算的偏差，就需要进行修改。因此，预算可以作为一种度量资源实际使用量和计划量之间差异的基线标准而使用。此外，成本预算在整个计划和实施过程中起着重要作用。预算与资源的使用相联系，通过成本预算掌握开发进度。在软件开发过程中，应不断收集和报告有关进度和费用数据，以及对未来问题和相应费用的预计，通过对比预算进行控制，必要时对预算进行修正。

2. 编制成本预算的原则

为了使成本预算真正发挥作用，编制成本预算时应遵循以下原则：

（1）成本预算要与软件开发目标相联系。开发目标包括质量目标和进度目标。成本与质量、进度关系密切，三者既统一又对立，进行成本预算确定成本控制目标时，必须同时考虑质量目标和进度目标。质量目标要求越高，成本预算也越高；进度越快，成本越高。因此，成本预算要与质量计划和进度计划保持平衡，防止顾此失彼，相互脱节。

（2）成本预算要以软件需求为基础。软件需求是成本预算的基石。如果以模糊的需求为基础进行成本预算，则预算不具有现实性，容易发生实际成本超支。

（3）成本预算要切实可行。成本预算过低，实际费用过高，或者实际费用过低，预算过高，都会失去成本控制基准的意义。所以编制成本预算时，要根据相关的财经法规、方针政策，从实际出发，既达到控制成本的目的，又切实可行。

（4）成本预算要有一定的弹性。在软件开发过程中，经常有难以预料的事件发生，这就会对预算的执行产生影响。因此，编制成本预算要留有一定的余地，使预算具有适应条件变化的能力，即具有一定的弹性。通常可以在整个项目预算中留出10%～15%的不可预见费用，以应付开发过程中可能出现的意外情况。

3. 编制成本预算的步骤

编制成本预算主要分两步进行：

（1）分摊总预算成本。分摊总成本到各成本要素（如人工、设备、分包商）中去，再落实到 WBS 中的工作包，并为每一个工作包建立总预算成本（Total Budgeted Cost，TBC）。为每一个工作包建立 TBC 的方法有两种：一种是自上而下法，即在总成本之内按照每一工作包的工作范围，按总成本的一定比例分摊到各个工作包中；另一种是自下而上法，依据每一个工作包的具体工作进行成本估计。为了成本预算，需要预测软件开发将要耗费何种资源、各种资源的需要量、未来通货膨胀的影响等。每一工作包的 TBC 是组成该工作包的所有活动的预算成本之和。

（2）累计预算成本。一旦为每一工作包建立了总预算成本，就要把总预算成本分配到整个工期的各阶段中去，每期的成本估计根据组成该工作包各个活动所完成的进度确定。累计预算成本（Cumulative Budgeted Cost，CBC）是一个合计数，是直到某期为止按进度完成的软件预算成本的累计值，是分析软件成本绩效的基准。

12.5 软件成本控制

成本控制是根据计划阶段对开发成本的估算和预算，在开发过程中定期进行成本考察，统计实际成本和预算成本的偏差，对未完工部分进行有效的成本控制，精确把握成本使用，以求最合理地使用开发经费，是减小实际成本与预算成本偏差的有效手段。

12.5.1 成本控制流程

软件项目的成功率非常低，软件开发有高度不可预测性。软件危机的主要原因之一就是对开发成本和进度的估算不准确、执行过程缺乏控制。软件开发成本控制是避免或最终根除软件危机的重要途径之一。

成本控制是在项目开发过程中，定期收集项目的实际成本数据，与成本的基准计划值进行对比分析，及时发现并纠正偏差，来控制项目预算的变化。其目的是将成本控制在预算范围内，或者控制在可接受的范围内，以便在项目失控前就及时采取措施予以纠正，以取得最合理的绩效。成本控制过程的输出是修正的成本估算、更新预算、纠正行动和取得的教训。成本控制是项目管理的重要内容，成本控制过程是一个动态过程，其流程如图12.8所示。

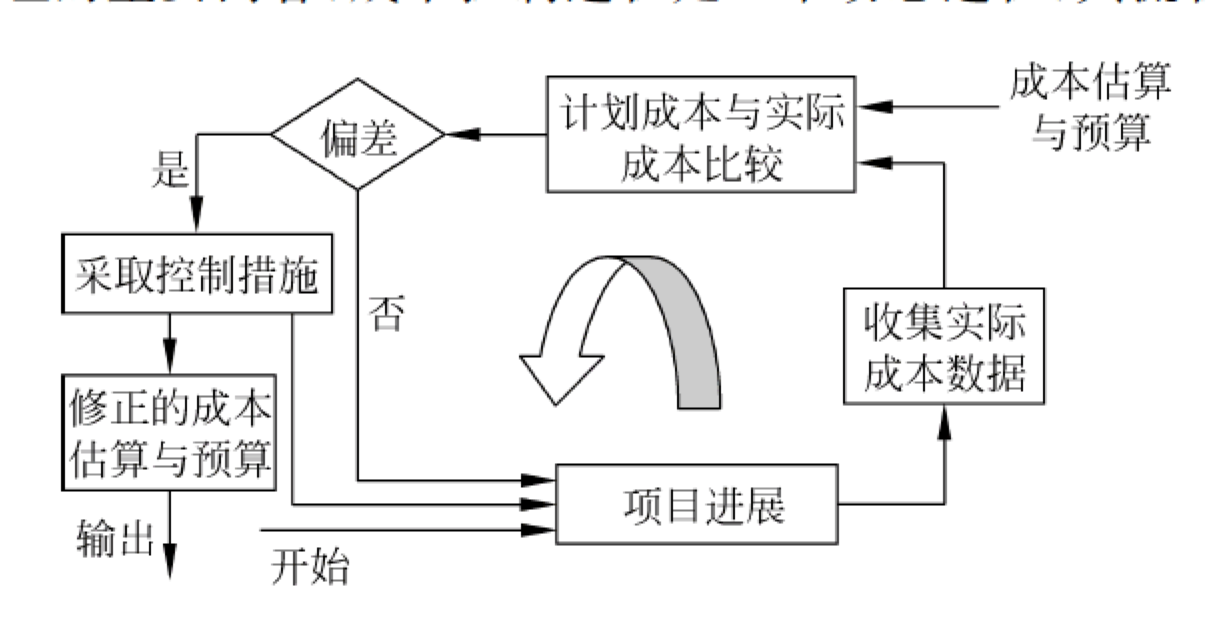


图12.8 动态成本控制流程

12.5.2 成本控制措施

软件成本控制要开源节流，即增收节支。只开源不节流或只节流不开源，都达不到好的效果。成本控制可采取以下措施：

(1) 组织措施。项目经理是第一责任人，负责全面的管理与控制；技术经理应尽可能采用先进技术，保证如期完成，降低实施成本；销售人员要使项目及时回款，最好使甲方追加投资；财务人员及时分析财务收支，做好资金调度；其他人员也要为增收节支尽职尽责。

(2) 技术措施。制定先进合理的实施方案，达到缩短工期、提高质量、降低成本的目的；寻求降低风险、提高效率和避免失败的新技术、新方法，以降低成本；严把质量关，杜绝返工现象发生，缩短验收时间，节省费用开支。

(3) 经济措施。经济措施是成本控制的核心，主要表现在以下几个方面：

① 人工费用控制管理。软件成本主要是人工费用。改善人员组织，减少因等待造成的窝工现象；实行合理的奖惩制度；加强技术教育和培训工作；加强劳动纪律，严格控制松紧不均的工作安排。

② 差旅费用控制管理。差旅费用是第二大成本开支，应尽量减少在用户现场的开支和滞留时间。可以通过加强需求分析、测试和事前沟通、出差前明确目标和任务等途径实现。

③ 设备费用控制管理。正确选配、合理利用计算机设备，提高设备使用效率，加快进度、降低设备使用费用。

④ 间接费用及其他直接费用控制管理。精简管理机构，合理确定管理幅度与管理层次，节约实施管理费用等。

项目成本控制的组织措施、技术措施、经济措施，三者是融为一体，相互作用的。项目经理是项目成本控制的主要负责人，要以合同为依据，制定项目成本控制目标，项目组内各业务经理要通力合作，形成以合同和预算为目标的项目实施方案经济优化、外部采购经济优化、人力资源配备经济优化的项目成本控制体系。

12.5.3 成本控制方法

挣值管理(Earned Value Management,EVM)是用与进度计划、成本预算、实际成本联系的三个独立变量进行项目绩效管理和成本控制的一种方法。挣值管理通过比较计划工作量、完成量(挣得)与实际成本花费,以测算成本和进度是否符合原定计划,进而对成本进行控制。

1. 挣值的概念

挣值(Earned Value,EV)是专门用来有效地度量和比较已完成作业量和计划要完成作业量的变量,是挣值管理分析方法的关键要素。由于软件用户根据该值对软件开发商完成的工作量进行支付,也就是开发商获得(挣得)的金额,故称挣值,也叫赢得值、挣得值。在项目管理过程中,当活动完成时,才能获得这部分价值。

挣值有以下三个特点:

(1) 无论是整个项目还是项目的一部分,度量单位都是统一的。传统使用的度量单位包括工时和资金。对劳动力密集的行业来说,工时通常被认为是充足的,而工程的费用由记账系统控制。当整个工程费用被工程控制系统控制时,把资金当做挣值度量单位标准是比较有效的。因为每个劳动工时是有价格的,所以费用也就同样能控制劳动。

(2) 挣值管理是分析工程计划和执行情况的一贯方式。如果询问不同行业人员工作的计划和执行情况,很可能得到不同的答案,这是因为不同行业计算计划和进度的方法不同。因此,如果需要明确工程项目的进展情况,就必须使用挣值来建立一种能确定计划进度和实际进度的特别方法。

(3) 挣值管理是分析工程项目成本执行效果的基础。如果想要知道在工程项目完成之前工程费用的实际支出情况,就需要知道任意时刻的计划成本以及已完成工作的实际费用。在传统的项目成本管理中,如果只知道某一时刻的计划成本和实际成本,但并不知道实际完成工作的计划成本,就很难确定当前成本是在计划内还是超出计划。这是挣值提供的遗漏信息。

2. 三个关键指标

(1) 计划工作量预算费用(Budgeted Cost for Work Scheduled,BCWS):指某阶段计划完成工作量所需的预算成本。

(2) 已完成工作量实际费用(Actual Cost for Work Performed,ACWP):指某阶段完成实际工作量所消耗的费用。

(3) 已完成工作量预算成本(Budgeted Cost for Work Performed,BCWP):即挣得值,指在某阶段实际完成工作的价值。

以上三个关键指标是关于时间的函数。函数曲线呈 S 形状,通常称为 S 曲线。

3. 四个评价指标

(1) 费用偏差(Cost Variance,CV):是检查时点上挣值(BCWP)与实际成本(ACWP)之间的差异。计算公式为:

$$CV = BCWP - ACWP \quad \text{（即：费用偏差 = 挣值 − 实际成本）} \tag{12.6}$$

图 12.9 为费用偏差分析图。CV＜0，表示执行效果不佳，实际消耗费用超过预算，即超支，如图 12.9(a)所示；CV＞0，实际消耗费用低于预算，即有节余或效率高，如图 12.9(b)所示；CV＝0，实际消耗费用等于预算值。

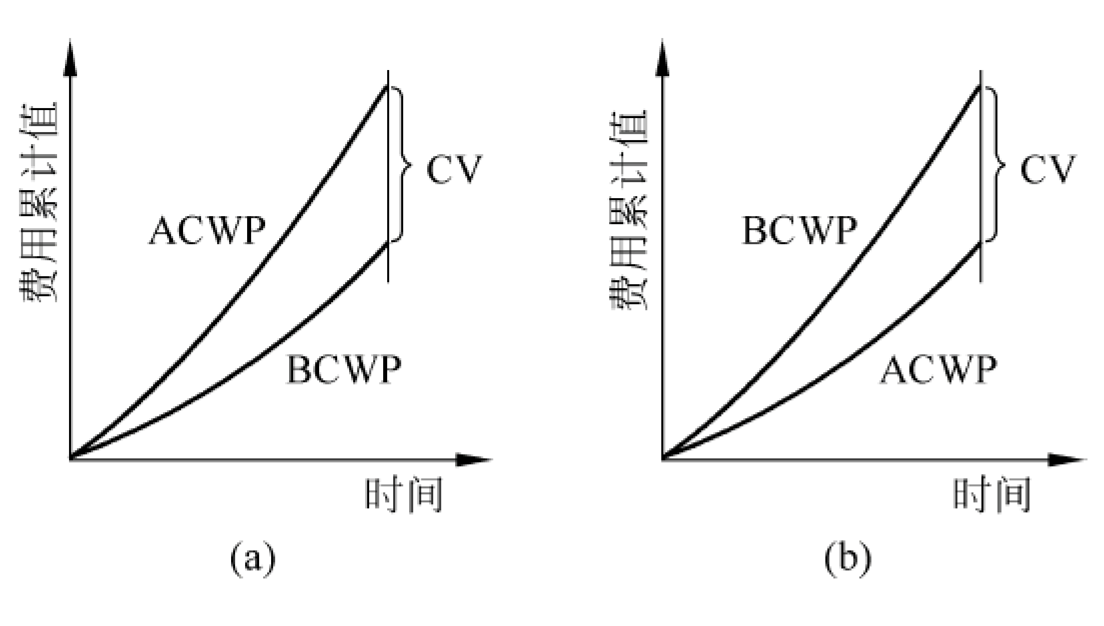


图 12.9　费用偏差分析图

(2) 进度偏差(Schedule Variance，SV)：是检查时点上挣值(BCWP)与计划成本(BCWS)之间的差异。计算公式为：

$$SV = BCWP - BCWS \quad \text{（即：进度偏差 = 净值 − 计划成本）} \tag{12.7}$$

图 12.10 为进度偏差分析图。SV＞0，表示进度提前，如图 12.10(a)所示；SV＜0，表示进度延误，如图 12.10(b)所示；SV＝0，表示实际进度与计划进度一致。

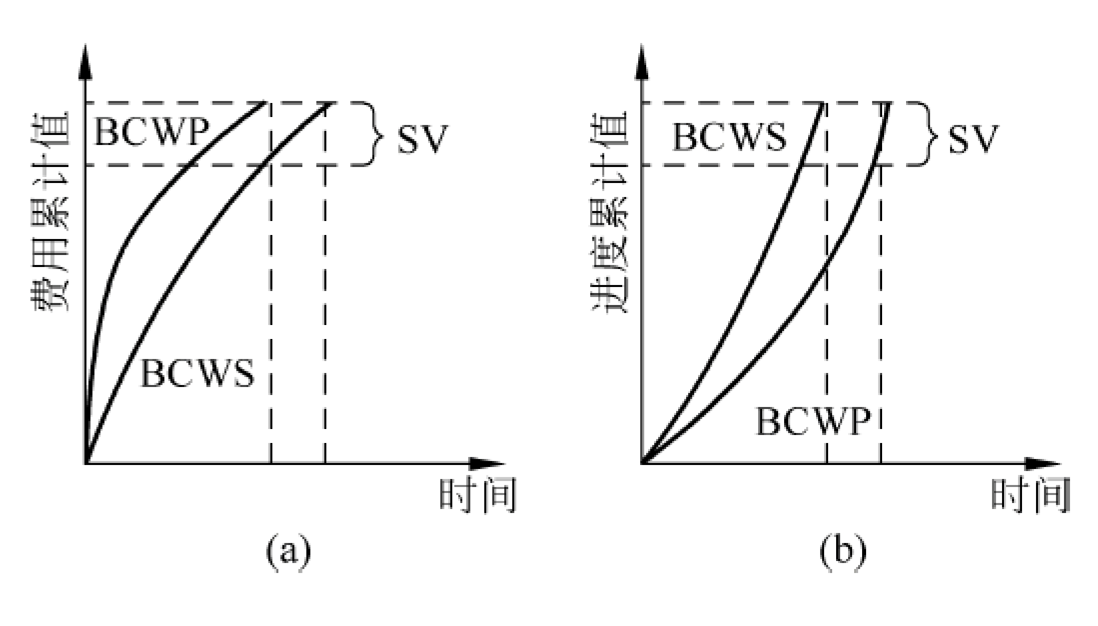


图 12.10　进度偏差分析图

(3) 费用执行指标(Cost Performed Index，CPI)：指预算费用与实际费用之比。计算公式为：

$$CPI = \frac{BCWP}{ACWP} \tag{12.8}$$

CPI＞1，表示低于预算，即实际费用低于预算费用；CPI＜1，表示高于预算，即实际费用高于预算费用；当 CPI＝1，表示实际费用与预算费用相符。

(4) 进度执行指标(Schedule Performed Index，SPI)：指挣得值与计划值之比。计算公式为：

$$SPI = \frac{BCWP}{BCWS} \tag{12.9}$$

SPI＞1，表示进度提前，即实际进度比计划进度快；SPI＜1，表示进度延误，即实际进度落后于计划进度；SPI＝1，表示实际进度与计划进度相符。

4. 用挣值曲线控制成本

将每期(根据项目大小,可以是周、旬、月等)的成本汇总分析表绘制成挣值曲线,如图 12.11 所示。每期的值都对应一个具体的检查日期,在项目未完工前图中没有完工日期或计划完工日期,但也能清楚地表示出各项指标值间的关系。BCWP(挣得值)曲线随时间的增长而增加,呈现一条 S 形曲线。BCWS 是计划成本值,ACWP 是实际费用值。

项目经理要关注曲线的变化(曲线交替,改变了 SV、CV 的值,使负数变成正数)的点。通过项目的检查点,考察 BCWP、BCWS 和 ACWP 三条曲线,SV、CV 两个指标,比较容易看出进度、费用与计划之间的关系。

图 12.11 中最上面的曲线 ACWP 是实际花费的费用,因为费用超支,它在计划成本 BCWS(第二条曲线)的上面;第二条曲线是计划成本,是一条“基准”线;第三条曲线 BCWP 是进度与费用挂钩的综合(挣得)曲线,也可以看成是一条“评价”曲线。图中进度和费用不理想,挣得曲线位于最下端。

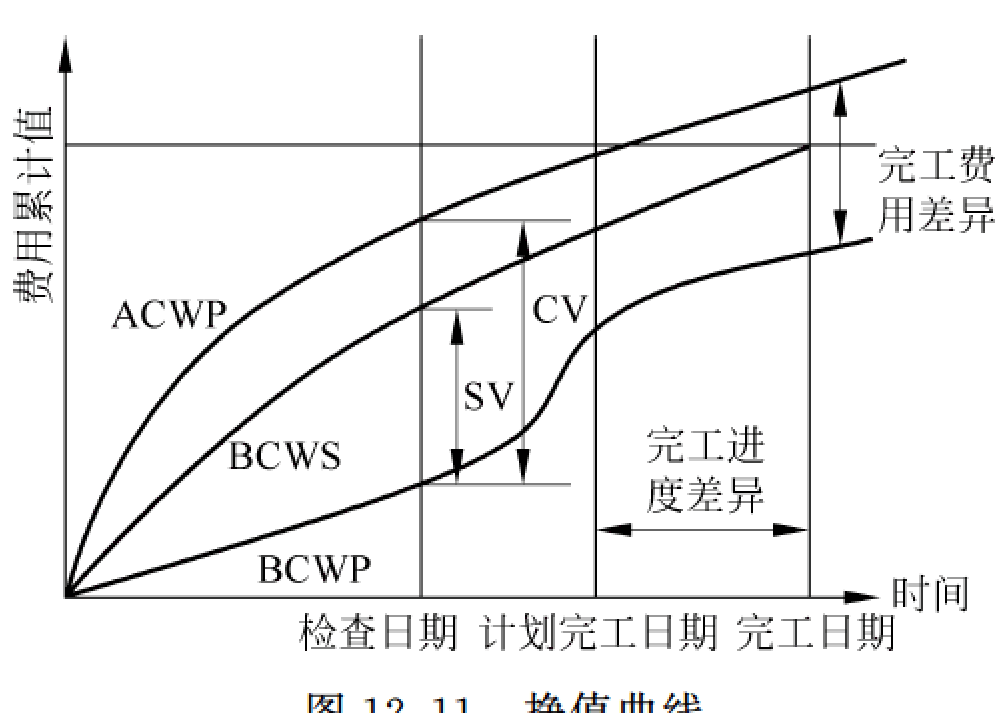


图 12.11 挣值曲线

思考题

1. 成本管理包括哪些过程?
2. 软件成本的特点是什么?
3. 软件成本主要由哪几个部分构成?
4. 如何理解质量对成本的影响?
5. 如何理解工期对成本的影响?
6. 简述资源计划编制步骤及各步的主要工作。
7. 常用的资源计划编制工具有哪些?
8. 功能点分析法需要哪几步完成软件成本估算?
9. 掌握功能点分析法估算软件成本的方法。
10. 成本预算的特征是什么?
11. 编制成本预算时应遵循哪些原则?
12. 理解成本控制流程。
13. 成本控制有哪些措施?
14. 如何运用挣值曲线控制成本?

第13章 配置管理

采用软件配置管理系统能够解决很多现实问题，对于程序员，可以安全地保护每天的劳动成果，同时对有关配置结构有比较清晰的概念，也可以获取到欲得到的配置信息；对于项目经理，能够方便地协调项目进展过程中各成员之间的工作，提高整个开发团队的协同工作能力；对于公司领导，可以了解整个组织的当前状态，对组织的全局实施控制，以保证产品及时交付给用户，并且对用户问题做出适当的反应。

随着软件系统的日益复杂化，以及用户需求、软件更新频繁化，软件配置管理逐渐成为软件生命周期的重要控制过程，是软件工程中质量管理的重要内容。

13.1 配置管理概述

软件配置管理(Software Configuration Management，SCM)是通过技术或行政手段，对软件产品及其开发过程和生命周期进行控制、规范的一系列措施，目的是记录软件产品的演化过程，最大限度地减少错误和混乱，保证软件项目工作产品在整个生命周期内的完整性。在本书的配置管理(Configuration Management，CM)就是指软件配置管理。

配置管理过程是对处于不断演化、完善过程中的软件产品的管理过程。其最终目标是实现软件产品的完整性、一致性、可控性，使产品最大程度地与用户需求相吻合。通过控制、记录、追踪对软件的修改和每个修改生成的软件组成部件，实现对软件产品的管理。

13.1.1 配置管理需求分析

现在的软件开发通常是许多人共同合作。在团队开发模式中，软件项目管理就显得更加重要，并直接影响到软件产品的质量。如果缺乏对软件开发过程的统一管理，产生的问题可通过图 13.1 表示。

缺乏统一管理出现的问题具体描述为：

(1) 由于开发经费及开发时间的限制，不可能一次开发就解决所有问题，许多问题有待维护阶段解决，由此带来软件产品的不断升级，而维护和升级必需的文档往往非常混乱。

(2) 开发过程缺乏规范化管理，即使有源程序以及相应的文档，也由于说明不详细而不能对产品进行功能扩充，用户不得不再次投入经费去开发新产品，浪费人力、物力和时间。

(3) 在团队式开发中，人员流动在所难免。如果管理不善，有些人员流动将对开发工作产生致命影响。特别是软件项目管理人员或核心技术人员流失，可能导致无法确定软件产

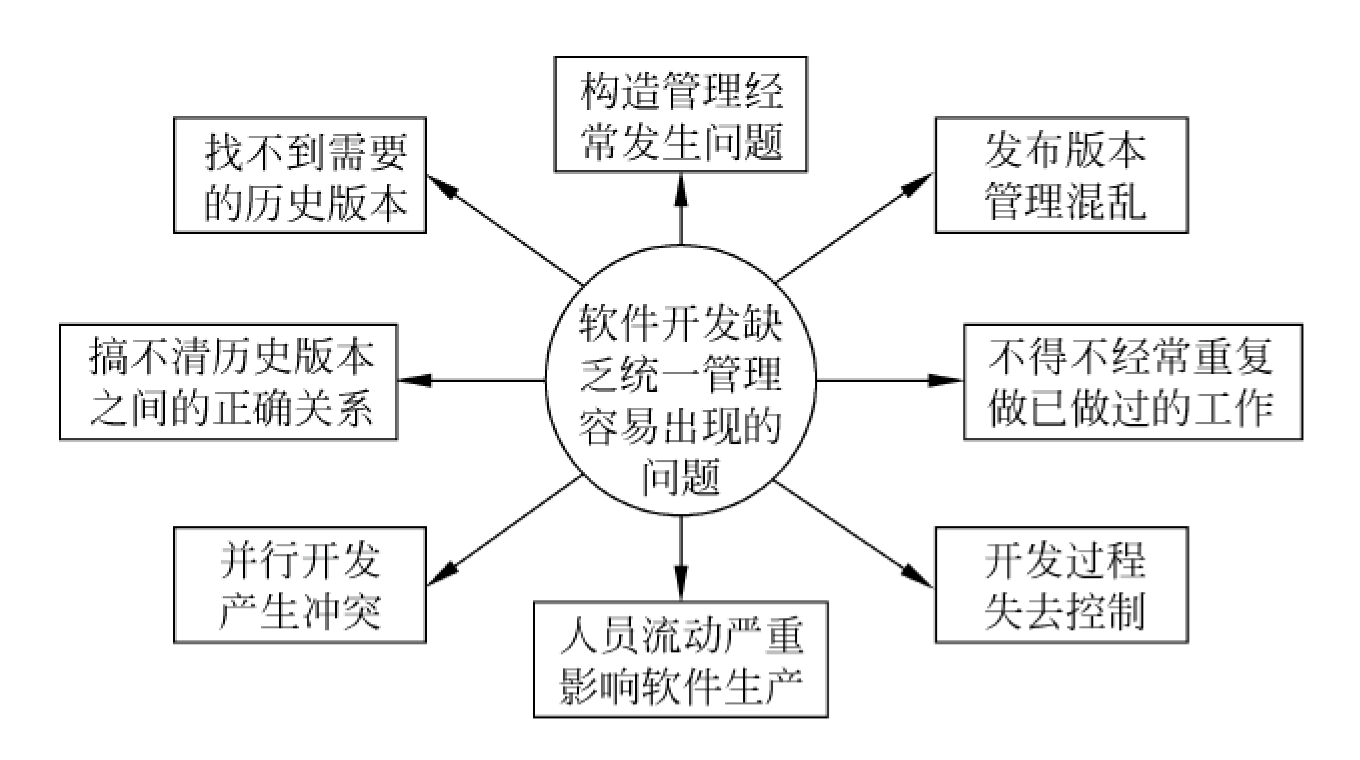


图 13.1 缺乏统一管理出现的问题

品中各模块所处的状态及阶段，使软件产品版本出现混乱，甚至泄露公司的核心机密。

(4) 管理不善可能导致未经测试的软件成分加入到产品中，不但影响产品质量，有时还会导致致命错误，造成不可挽回的损失。

(5) 用户利益无法保证，用户与开发商缺乏有效的沟通手段。用户投入了开发费用后，通常得到的只是可执行文件。即使是较好的文档，对不熟悉开发过程的非专业人员来说也无从下手，更谈不上日后的维护和升级。

(6) 软件生产达不到规模化，无法形成软件企业的内部标准构件仓库。软件产品总处于一种低水平、重复开发的状态，不但时间得不到保证，而且成本也无法降低，产品缺乏市场竞争力。

这些问题在实际开发中表现为项目组成员沟通困难、软件重用率低下、开发人员各自为政、代码冗余度高、文档不健全。由此造成的后果是数据丢失、开发周期漫长、产品可靠性差、软件维护困难、用户抱怨使用不便、项目风险增加。

13.1.2 配置管理的作用

配置管理在软件开发过程中越来越重要。一个好的配置管理过程能覆盖软件开发和维护的各个方面，同时对软件开发过程的宏观管理，即项目管理，也有重要的支持作用。良好的配置管理能使软件开发过程有更好的可预测性，使软件系统具有可重复性，使用户和主管部门对软件质量和开发小组有更强的信心。

好的配置管理过程有助于规范各个角色的行为，同时又为角色之间的任务传递提供无缝接合，使整个开发团队高效率地协同工作。配置管理过程所规范的工作流程和明确分工，有利于管理者应付开发人员流动的困境，减少因人员流动造成的损失。

配置管理对软件开发项目的具体作用可表现在以下几个方面：

(1) 缩短开发周期。通过配置管理工具对程序资源进行版本管理和跟踪，建立公司的代码知识库，保存开发过程中每个过程的版本，这样可以提高代码重用率，便于同时维护多个版本和进行新版本开发，防止系统崩溃，最大限度地共享代码。同时，项目管理人员通过查看项目开发日志对开发过程进行管理，测试人员可以根据开发日志的不同版本对软件进行测试，工程人员可以得到不同的运行版本。有些配置管理工具还提供 Web 版本，供外地实施人员存取最新版本，无须开发人员亲临现场。

(2) 减少施工费用。利用配置管理工具,建立开发管理规范,把版本管理档案链接到公司内部的 Web 服务器上,内部人员可直接通过 IE 访问,工程人员通过远程进入内部网,获取所需的最新版本。开发人员无须亲自到现场,现场工程人员收集反馈意见,书面提交给公司内部的开发组项目经理,开发组内部讨论决定是否修改。这样可以避免开发人员将时间和精力浪费在旅途中,同时节约差旅费用。

(3) 代码对象库的建立。软件代码是软件开发人员脑力劳动的结晶,也是软件公司的宝贵财富,长期开发过程中形成的各种代码对象就如同一个个已生产好的标准件一样,是快速生成系统的组成部分。没有配置管理,一旦某个开发人员离开工作岗位,其原来的代码便基本无人过问。究其原因,就是没有对开发人员的代码进行管理,没有把使用范围扩大到公司一级,没有进行规范化。配置管理对软件对象管理提供了平台和仓库,有利于建立公司级的代码对象库。

(4) 建立业务及经验库。通过配置管理,可以形成完整的开发日志及问题集合,以文档方式伴随开发的全过程,不以某个人的转移而消失,有利于公司积累业务经验,无论对版本修改还是版本升级,都具有重要的指导作用。

(5) 量化工作量考核。在传统的开发管理中,工作量一直是难以估量的指标,靠开发人员自己把握,随意性很大;靠管理人员把握,主观性又太强。采用配置管理工具,开发人员每天下班前对修改的文件上传,记录当天修改的细节,这些描述可以作为工作量的衡量指标。

(6) 规范测试。采用配置管理,测试工作有了实实在在的内容,测试人员根据每天的修改细节描述,对每天的工作进行测试,对测试人员也具有可考核性,这样环环相扣,减少了工作的随意性。

(7)加强协调与沟通。采用配置管理,通过文档共享及其特定锁定机制,为项目组成员之间搭建交流的平台,加强了项目组成员之间的沟通,做到有问题及时发现、及时修改、及时通知,但又不额外增加很多的工作量。

从这些作用可以看出,配置管理确实能够解决困扰软件项目经理的很多问题。

13.2 配置管理的相关概念

13.2.1 软件配置项

软件配置是指一个软件产品在软件生命周期各个阶段产生的各种形式和各种版本的文档、程序及其数据的集合。而软件配置项(Software Configuration Item,SCI)就是该集合中的一个元素,可以是与合同、计划、开发过程、软件产品等有关的文档、数据、源程序、目标代码、可执行代码,也可以是软件开发工具、管理工具、第三方代码等。软件配置项是软件配置管理的对象。

1. 软件配置项的状态

在软件生命周期中,一般包括制定计划、分析评估、设计、测试、运行维护等状态,与此相对应,软件配置项也可划分为设计态、测试态、受控态和运行态四种状态。各状态之间的联系如图 13.2 所示。

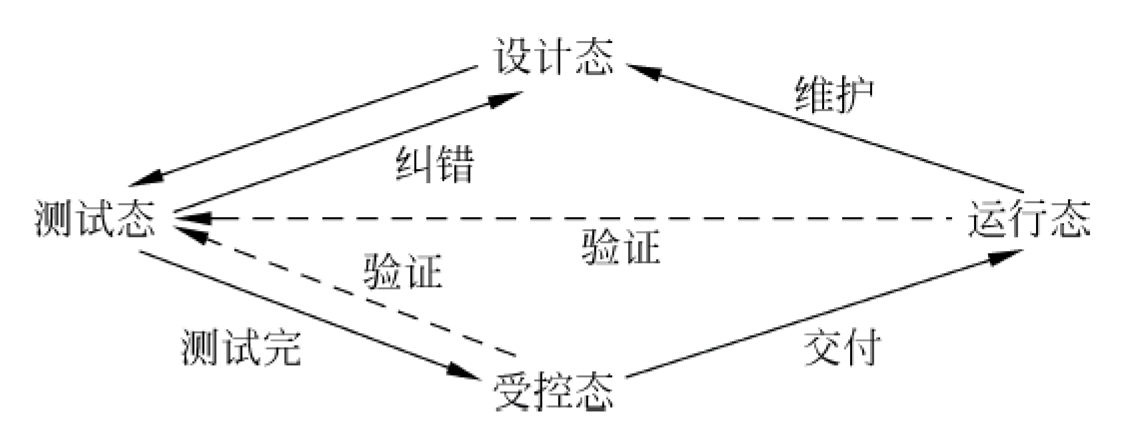


图 13.2 软件配置项的四种状态

这四种状态相互之间的联系具有方向性，沿图中实线箭头所指方向的状态变化是允许的，虚线表示为了验证或检测某些功能或性能而重新执行相应的测试，一般不沿虚线变化。

2. 软件配置项的版本

软件配置项也有不同的版本，配置项和配置项的版本类似于面向对象的类和实例。配置项可以看成是类，版本看成类的实例。图 13.3 表示的是数据库设计说明配置项。数据库设计说明的不同版本对应于数据库设计说明的实例；配置项的不同版本是从最原始的配置项(相当于配置项类)逐渐演变而来的，尽管每个版本都不相同，但具有相关性。



图 13.3 软件配置项类及实例

3. 软件配置项的分类

在软件开发过程中，最早的软件配置项是系统需求规格说明书。随着软件开发过程不断深入，需要纳入管理的各种工作产品越来越多，软件配置项的数量也会上升，而软件配置管理的目的是在软件项目的整个生命周期内建立和标识软件配置项，并对其进行控制管理、跟踪维护，保证其完整性和一致性。这样，软件配置管理的作用将清晰可见。通过对软件配置项进行分类，可以加强对软件配置项的管理。软件配置项的分类如表 13.1 所示。

表 13.1 软件配置项的分类

类别	特　征	实　例
环境类	软件开发环境及软件维护环境	编译器、操作系统、编辑器、数据库管理系统、开发工具、测试工具、项目管理工具、文档编辑工具
定义类	需求分析及定义阶段完成后得到的工作产品	需求规格说明书、项目开发计划、设计标准或设计准则、验收测试计划
设计类	设计阶段结束后得到的工作产品	系统设计规格说明、程序规格说明、数据库设计、编码标准、用户界面标准、测试标准、系统测试计划、用户手册
编码类	编码及单元测试后得到的工作产品	源代码、目标码、单元测试数据及单元测试结果
测试类	系统测试完成后的工作产品	系统测试数据、系统测试结果、操作手册、安装手册
维护类	进入维护阶段以后产生的工作产品	以上任何需要变更的软件配置项

13.2.2 基线

软件开发过程是一个不断变化的过程，由于各种原因，可能变动需求、预算、进度和设计方案等。尽管这些变动请求绝大部分是合理的，但在不同时期所做的不同变动，其难易程度和对成本的影响不同。为了有效地控制变动，软件配置管理引入了基线（base line）这一概念。

1. 基线的定义

IEEE 对基线的定义是："已经正式通过复审和批准的某规约或产品，它因此可作为进一步开发的基础，并且只能通过正式的变化控制过程改变。"根据这个定义，可以把开发流程中所有需要加以控制的配置项分为基线配置项和非基线控制项两类。基线配置项包括所有的设计文档和源程序；非基线控制项包括各类计划和报告。

基线是软件文档或源码（或其他产出物）的一个稳定版本，是进一步开发的基础。当基线形成后，配置管理人员需要通知相关人员，并且发布基线的版本。基线是项目储存库中每个版本在特定时期的"快照"。它提供正式标准，随后的工作基于此标准，并且只有经过授权后才能变更这个标准。建立一个初始基线后，以后每次对其进行的变更都将记录为一个差值，直到建成下一个基线。

参与项目的开发人员将基线代表的各版本的目录和文件填入工作区。随着工作的进展，基线将合并自从上次建立基线以来开发人员已经交付的工作。变更一旦并入基线，开发人员就采用新的基线，与项目中的变更保持同步。调整基线将把集成工作区中的文件并入开发工作区。

虽然基线可以在任何级别上定义，但一般最常用的软件项目基线如图 13.4 所示。基线提供了软件生命周期中各个开发阶段的一个特定点，作用是把开发阶段的工作划分得更加明确化，使本来连续的工作在这些点上断开，便于检查与肯定阶段成果。在交付项中确定一致的子集，作为软件项目配置基线，这些版本一般不是同一时间产生的，但具有在开发的某一特定步骤上相互一致的性质，例如系统的一致、状态的一致。基线可以作为一个检查点，正式发行的产品必须是经过控制的基线产品。

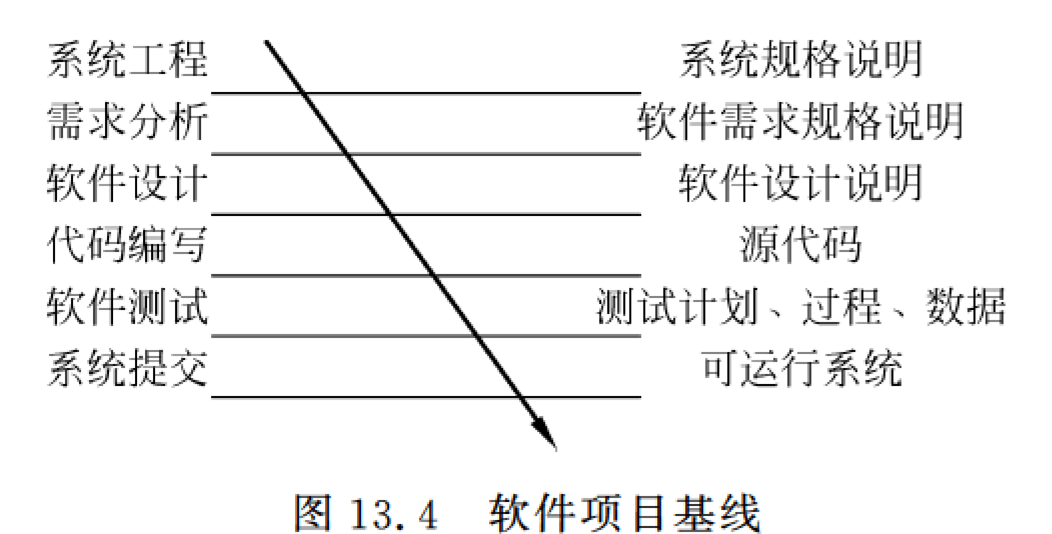


图 13.4 软件项目基线

2. 建立基线的原因

建立基线的原因如下：

（1）重现性。重现性是指及时返回并重新生成软件系统给定发布版本的能力，或者是在项目中的早期重新生成开发环境的能力。

(2) 可追踪性。可追踪性是指建立项目部件之间的前后继承关系，其目的在于确保设计满足要求、代码实施设计以及用正确代码编译可执行文件。

(3) 报告。报告来源于一个基线内容同另一个基线内容的比较，基线比较有助于调试并生成发布说明。

3. 建立基线的优点

建立基线有以下优点：

(1) 基线为开发部件提供了一个定点和快照。

(2) 新项目可以从基线提供的定点处建立。

(3) 各开发人员可以将建有基线的构件作为在隔离的私有工作区中进行更新的基础。

(4) 当认为更新不稳定或不可信时，基线为团队提供一种取消变更的方法。

(5) 利用基线重新建立基于某个特定发布版本的配置，可以重现已报告的错误。

(6) 定期建立基线，以确保各开发人员的工作保持同步。

13.2.3 版本

版本是某一配置项已标识了的实例。版本用来定义一个具体实例应该具有什么样的内容和属性。随着软件的开发进展，版本也在不断地演变，这些不同配置项的不同版本构成了一个复杂的版本空间。

一个系统版本就是一个系统实例，在某种程度上有别于其他系统实例。系统新版本可能有不同的功能、性能，可能修改了系统错误。有些版本可能在功能上没有什么不同，只是为不同的硬件或软件配置而设计的。如果版本之间只有细微区别，有时就把其中的一个版本称做另一个版本的变体。

一个系统的发布版本是要分发给用户的版本。每个系统发布版本都应该包含新的功能或是针对不同的硬件平台。一个系统的版本要比发布版本多得多，因为机构的内部版本是为内部开发或测试而创建的，有些根本不会发布到用户手中。

版本的演变一般有两种方式：串行演变和并行演变。串行演变所形成的每一个新版本都是由当前最新版本演变而来的。这样，各个不同版本按演变过程就形成了一个简单链，称为版本链。在这种方式下，版本的演变是按照一对一的映射关系前进的，通常是为了弥补缺陷、提高性能或适应环境。并行演变采用一对多的方式进行。在实际应用中，这两种版本演变形式通常结合在一起，形成更为普通、带分支的版本图，也称为版本树。版本树反映了项目开发演变的历史。版本树示例如图 13.5 所示。

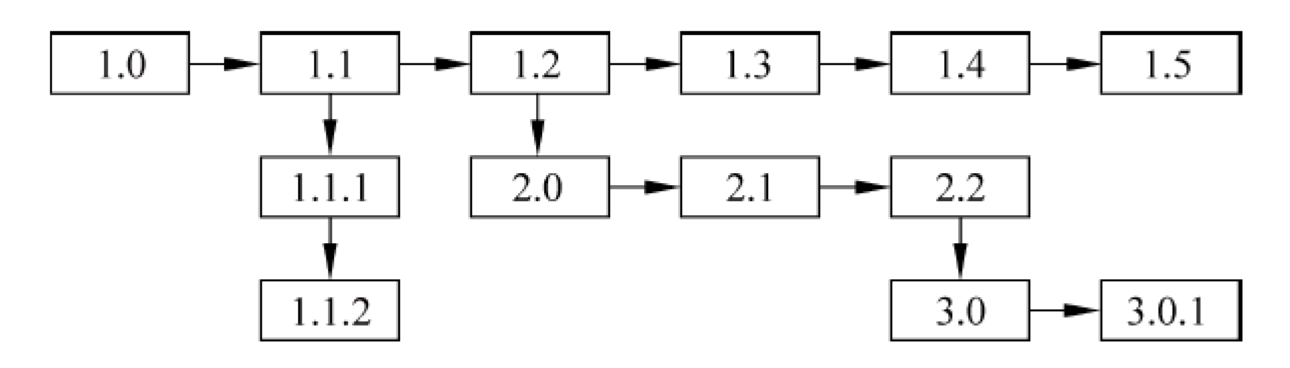


图 13.5 版本树示例

13.2.4 配置数据库

配置数据库(Configuration Management DataBase,CMDB)用于记录与配置有关的所有信息,帮助评估因系统变更带来的影响,并提供有关配置管理过程的管理信息。除了定义配置数据库的模式以外,还要定义记录和检索项目信息的规程,这是配置管理规划过程的一部分。

配置数据库不仅包含有关配置项的信息,可能也包含组件用户、系统用户、可执行平台及计划变更等信息。配置数据库必须能够对各种系统配置查询做出应答。

在理想情况下,配置数据库与版本管理系统集成到一起,版本管理系统负责存储和管理正式项目文档。这种方法(某些集成 CASE 工具支持该方法)使得变更与受变更影响的文档和组件间的直接链接成为可能。例如,设计文档和程序代码之间的链接能得到维护,这样在提出一个变更时,可以较容易地找出所有必须修改的地方。

配置数据库存储配置项的有关信息并在版本管理系统或文件存储中索引它们的名字。虽然这种做法费用低廉、使用灵活,但缺点是配置项的变更可能不经过配置数据库。因此,不能确定配置数据库是否反映了系统的最新状态。

13.3 配置管理的组织

在典型的软件开发项目中,配置管理的组织大多数是由管理层和职能层共同组成的。其组织结构如图 13.6 所示。

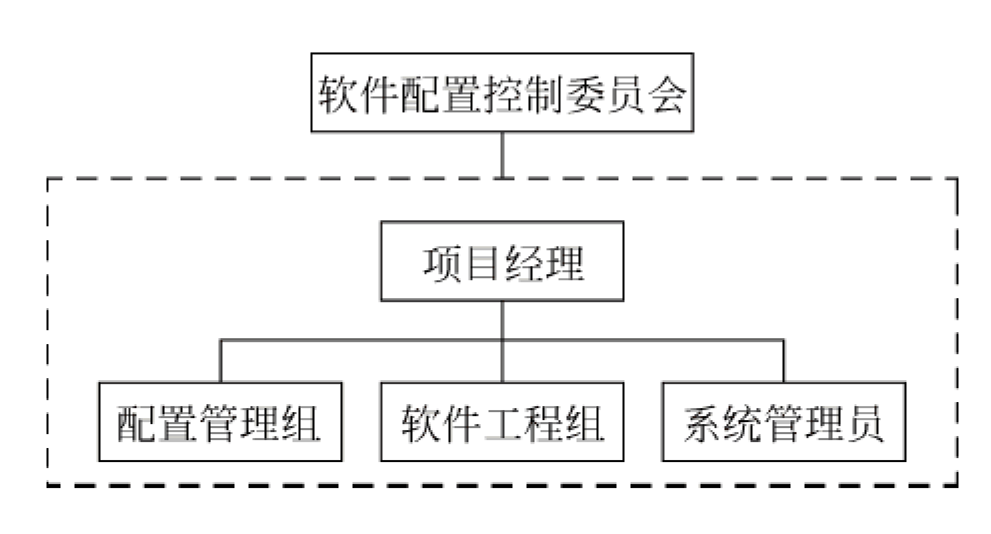


图 13.6 配置管理的组织结构

1. 软件配置控制委员会(Software Configuration Control Board,SCCB)

在项目开始时,由项目经理根据项目情况确定委员会人员构成,并记录在配置管理计划中。软件配置控制委员会一般由项目经理、技术专家、业务专家构成。软件配置控制委员会负责指导和控制配置管理的各项具体活动的进行,负责一些日常配置管理事务的决策,并在重大问题上为项目经理的决策提供建议。软件配置控制委员会在软件配置管理活动中出现的各种不能解决问题的报告链中处于最高点,拥有最高的权力,有权最终处理 SCM 问题。其具体职责如下:

(1) 协调组织一级的软件配置管理活动。

(2) 负责审核软件配置项的完整性、合理性,并拥有最终的授权。

(3) 负责分析需求变更的影响范围。

(4) 审查批准需求基线变更请求。

（5）审核软件配置管理活动。

（6）制定软件配置项出库与入库的控制规范。

2. 项目经理

项目经理是整个软件开发活动的负责人，在配置管理活动中，其主要工作是协调软件配置控制委员会和配置管理其他组织之间的工作，根据软件配置控制委员会的建议，批准配置管理的各项活动，控制配置管理的进程。其具体职责如下：

（1）批准配置管理计划。

（2）决定项目起始基线和开发里程碑。

（3）决定配置管理员。

（4）参与产品配置库的创建以及项目配置项结构的建立。

（5）审查基线审核报告以及对提出的问题制定纠正措施。

（6）接受并审阅配置控制委员会的报告。

3. 配置管理组（Software Configuration Management Group，SCMG）

配置管理组全面负责执行软件配置管理活动。其具体职责如下：

（1）负责配置库的创建、授权和维护。

（2）标识软件配置项。

（3）执行版本控制和变更控制方案。

（4）定义各角色的权限。

（5）撰写软件配置的状态报告。

（6）编写和维护配置管理计划。

（7）基线发布，通知相关人员已发生的变更。

（8）完成配置审计并提交报告。

（9）负责配置库的备份和恢复。

（10）对软件工程人员进行相关的培训。

4. 系统管理员

系统管理员负责建立和维护软件配置管理活动的软件和硬件支持环境，维护系统安全，协助其他角色的工作，负责配置管理工具的应用试验。

5. 软件工程组（Software Engineering Group，SEG）

软件工程组负责软件开发，是软件配置管理中检入检出的主要执行者；负责变更的提出及参与分析；负责记录和跟踪已修改的问题。

13.4 配置管理的主要活动

在项目管理过程中，需要解决诸如采用何种方式标识和管理已存在程序的各种版本，在软件交付用户前后如何控制变更，利用什么办法来估计变更引起的各种问题等，这些问题可

归结到软件配置管理中的活动方面。软件配置管理包括配置标识、版本控制、变更控制、状态报告和配置审核五个主要活动，如图 13.7 所示。

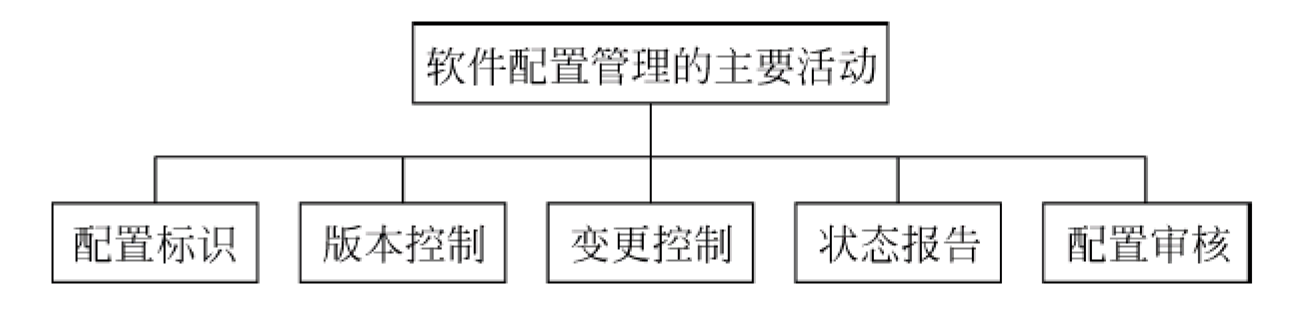


图 13.7　软件配置管理的主要活动

(1) 配置标识：在系统演化过程中表示中间的软件产品。
(2) 版本控制：记录每个配置项的发展历史，并控制基线的生成。
(3) 变更控制：在整个生命周期中控制中间软件产品的变化。
(4) 状态报告：记录和报告软件的变化过程。
(5) 配置审核：用于保证软件产品是依照需求、标准和合同开发出来的。

13.4.1　配置标识

配置标识(Configuration Identity，CI)是软件生命周期中划分选择各类配置项、定义配置项的种类、为它们分配标识符的过程。在软件开发过程中，随着软件生命期的向前推进，产生的配置项越来越多，为了控制和管理方便，所有的软件配置项都应该按照一定的方式来命名和组织，这是进行软件配置管理的基础。

每个软件配置项都包括名字、描述、资源列表和实际存在体四个部分。此外，在对配置项进行标识时，还要考虑配置项之间的关系。

配置标识管理是一个配置项的选择、命名和描述的过程。首先，把一个软件系统分成便于进行配置管理的各个配置项；接着，按照一定的方法对这些配置项命名、编号，以便于管理人员和开发人员明确该系统各配置项之间的相互关系，包括各个文档之间以及文档与代码之间的联系；最后，对每一个组成部件的功能、性能和物理特性进行必要的描述。

1. 配置标识的活动

配置标识的活动包括以下内容：

(1) 选择配置项。配置项是配置管理的最小单元，一般由一个或多个文件组成。组织可以根据不同的原则选择配置项。

(2) 制定配置项标识方案。选择好配置项后就要为其选择适当的标识方案。配置项的标识使配置项被唯一识别，并且标识方案可以显示软件演进的层次结构和追溯性。

(3) 制定存取方案。组织需要建立软件配置库，存放软件配置。这个配置库应使软件项目组的所有成员都可根据权限存取其中的配置项，同时必须协调各成员之间的关系，使每个成员所能执行的权限不超过其应有的范围。

2. 配置标识的对象

配置标识的对象包括如下：

(1) 各种功能规格说明和技术规格说明，以及软件项目的特殊功能和开发过程中使用

的方法。

(2) 所有受到功能和技术规格影响的开发工具，这些工具不仅包括用于创建应用程序的开发工具，而且还包括对比、调试和图形化工具。

(3) 所有与其他软件项目和硬件的接口。

(4) 所有与软件项目相关的文档和计算机文件，如文本文件、源程序、文档和图形，以及任意的二进制文件。

标识软件项不仅需要处理程序项和需求之间的联系，一般来讲，还需使用多种方式标识软件项，以及软件项同软件产品之间的关联。

3. 配置标识实例

表示方法："项目名称-所属阶段-产品名称-版本号"。

其中，版本号的约定如下：以"V"开头，版本号可分为三小节，即主版本号、次版本号和内部版本号，每小节以"."间隔。

例如，"教务管理系统-软件设计-详细设计说明书-V2.2.1"。

如果项目名称或所属阶段用汉字表示，会使配置标识过长，可采用简写的数字或拼音代码。例如，教务管理系统用"EMS"表示。

13.4.2 版本控制

版本控制是软件配置管理的重要功能，负责为配置库中的所有元素自动分配版本标识，并保证版本命名的唯一性。版本控制的目的是便于对版本变化加以区分、检索和跟踪，以表明各个版本之间的关系。一个版本是软件系统的一个实例，在功能和性能方面与其他版本有所不同，或是修正、补充了前一个版本的某些不足。

版本控制包括检入检出控制、分支和合并、历史记录。

1. 检入检出控制

软件开发人员对源文件的修改不能在软件配置管理库中进行，而是依赖于基本的文件系统并在各自的工作空间上进行。为了方便软件开发，需要不同的软件开发人员组织各自的工作空间。一般来说，不同的工作空间由不同的目录表示，而对工作空间的访问由文件系统提供的文件访问权限加以控制。

访问控制需要管理各个人员存取或修改一个特定软件配置对象的权限。开发人员能够从库中取出对应项目的配置项进行修改，并检入到软件配置库中，对版本进行"升级"；配置管理人员可以确定多余配置项并删除。

同步控制的实质是版本的检入检出控制。检入是把软件配置项从用户的工作环境存入软件配置库的过程；检出是把软件配置项从软件配置库中取出的过程。检入是检出的逆过程。同步控制可用来确保由不同的人并发执行的修改不会产生混乱。

2. 分支和合并

版本分支(以一个已有分支的特定版本为起点，但是独立发展的版本序列)的人工方法是从主版本即主干上复制一份，并做上标记。在实行了版本控制后，版本的分支也是一份拷

贝，这时的复制过程和标记动作由版本控制系统完成。版本合并(来自不同分支的两个版本合并为其中一个分支的新版本)有两种途径：一种是将版本 A 的内容附加到版本 B 中；另一种是合并版本 A 和版本 B 的内容，形成新的版本 C。

对文件来说，分支与合并的结果就是形成具有图结构的版本历史，即版本图，如图 13.8 所示。

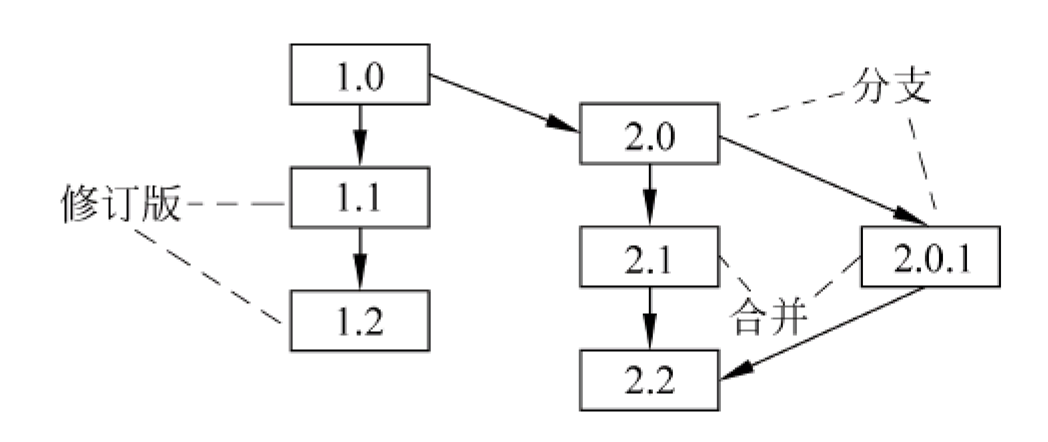


图 13.8 版本的分支与合并

版本分支的目的有以下几个：

(1) 代表独立的开发路径，如开发过程和维护过程。

(2) 代表组件的不同变体。

(3) 代表实验性的开发，该分支在以后可能被丢弃，或者被合并到主分支中。

(4) 适应两个开发人员并发地修改同一组件的情况。此时分支仅暂时存在，一旦两个修改完毕后将被合并。

合并就是将独立发生在两个版本分支上文件的修改合成到其中一个分支上，从而形成该分支上的一个新版本。合并包含两方面的内容：第一，两个文件版本内容的实际合并；第二，在版本图上作为版本历史的一部分进行反映。

3. 历史记录

版本的历史记录有助于对软件配置项进行审核，有助于追踪问题的来源。历史记录包括版本号、版本修改时间、版本修改者、版本修改描述等最基本的内容，还可以有其他一些辅助性内容，比如版本的文件大小和读写属性。

13.4.3 变更控制

变更控制是指在整个软件生命周期中控制软件的变化，建立一套对软件配置项的修改进行有意识的控制的机制，防止在软件开发过程中因盲目修改造成的混乱，主要是进行基线管理以及对基线更改控制过程的处理。

软件生命周期内全部的软件配置是产品的真正代表，必须使其保持正确。软件工程过程中某一阶段的变更，均要同时进行软件配置的变更，这种变更必须要严格加以控制和管理，保持修改信息，并把精确、清晰的信息传递到软件工程过程的下一步骤。变更控制是软件配置管理的核心，通过创建产品基线，在产品生命周期中控制基线的发布和变更。变更控制的目的是建立一个保证生产、符合质量标准的软件，保证每个版本的软件包含所有必要的元素，以及工作在同一版本中的各元素可以一起正常工作的机制。

1. 变更的波及面

变更是不可避免的，也是必不可少的。变更和变更控制是矛盾的统一体。由于变更的内容、变更的幅度都会直接影响到整个项目，所以时时刻刻需要考虑到变更的波及面。在一个瀑布模型的生命周期中，变更的波及面如图 13.9 所示。

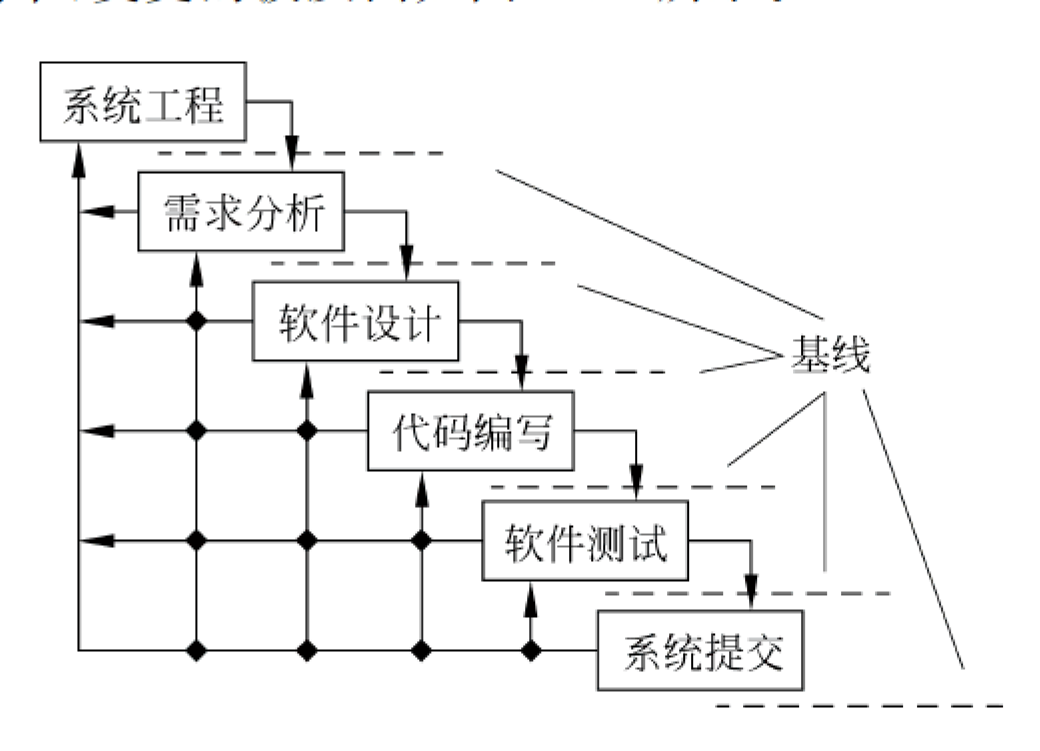


图 13.9 变更的波及面

在图 13.9 中，如果在系统工程阶段的工作需要发生变化，那么软件生命周期中的各个阶段都有可能受到影响，这些阶段所有相关的软件配置项也都会或多或少受到影响；如果代码编写阶段的工作需要变化，则软件测试阶段和系统提交阶段的工作都要受到影响；如果在代码编写阶段发现错误，该错误是由需求分析阶段的工作造成的，则需求分析及以下阶段都要进行相应的修改。

变更控制需要记录每次变化的相关信息，查看这些相关信息有助于追踪出现的各种问题。记录正在执行的变化信息有助于做出正确的管理决策。软件配置管理对基线管理的任务之一是追踪更改的变化过程，以保证对软件开发过程的可见性和可追踪性。对基线更改变化过程的追踪有两部分内容：一个是对基线更改版本的跟踪；另一个是对其更改原因以及更改结果的追踪。

2. 变更的种类

软件的变更通常有以下两种类型：

(1) 功能变更。功能变更是为了增加或者删除某些功能，或者为了改变完成某个功能的方法而进行的变更。这类变更如果代价比较小，对软件系统其他部分没有影响或者影响很小，通常应批准这类变更。反之，如果变更的代价比较大，或者对软件系统其他部分影响比较大，必须权衡利弊，以决定是否进行这类变更。

(2) 错误修补变更。错误修补变更是为了修复漏洞而要进行的变更，是必须进行的，通常不需要从管理角度对这类变更进行审查和批准。但是如果发现错误阶段在造成错误阶段的后面，例如，在实现阶段发现了设计错误，则必须遵照标准的变更控制过程，把变更正式记入文档，对所有受变更影响的文档都作相应的修改。

3. 变更的控制过程

为了防止开发人员对软件的随意变更，任何变更都需要经过严格的控制，必须经过变更

请求、变更评估、变更批准或拒绝、变更实现四个过程。其控制流程如图13.10所示。

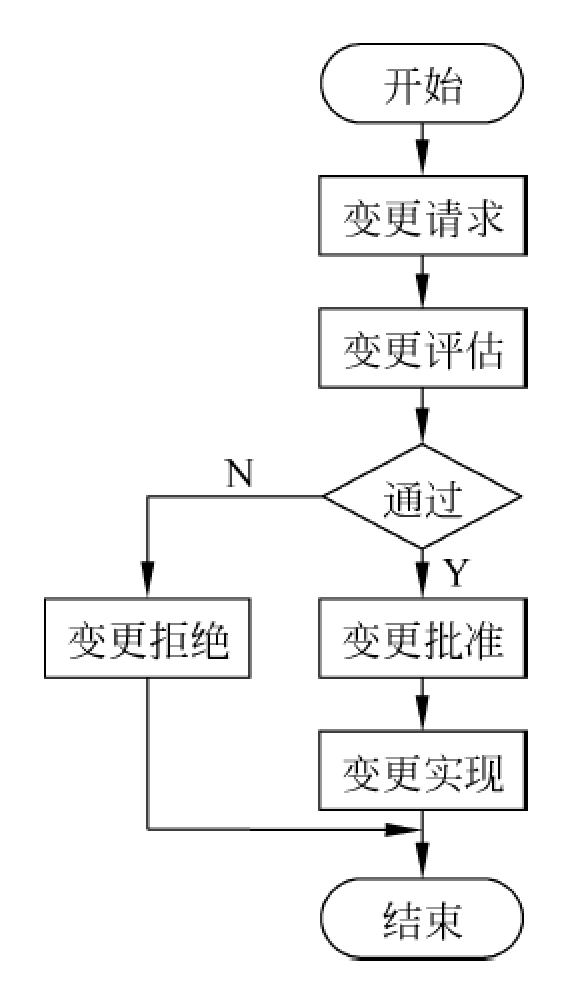


图13.10 变更的控制过程

(1) 变更请求。变更请求是变更控制的起始点,很少来自配置管理活动本身,通常由来自系统之外的事件触发。变更请求人确定变更是否需要进行,填写变更请求表,审核确定准确无误后,提交给项目经理。

(2) 变更评估。项目经理验证变更申请表的完整性、正确性和清晰性,对变更申请表进行初步分析,根据成本/效益和涉及的技术等因素,判断变更实施的必要性并估计变更方案,若初步判定变更请求可以接受,将此表送交配置控制委员会。配置控制委员会组织相关人员对变更进行评估。配置控制委员会对变更的评估内容如图13.11所示。

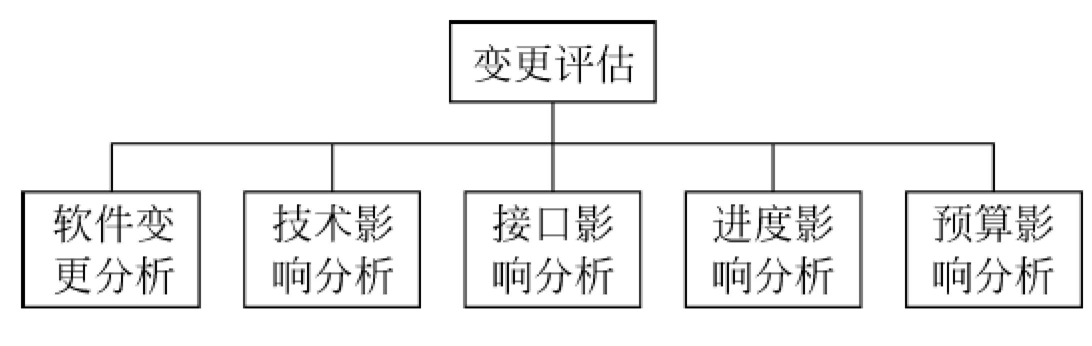


图13.11 变更评估内容

变更请求可能是由于请求人的错误理解而产生的,也可能与现存的请求相重复,如果在评估中发现提出的变更是不完整的、无效的或者已经评估的,就拒绝这一请求,并建立拒绝原因的文档,返回给提交变更的请求人员。

功能变更涉及软件额外费用问题。如果是功能变更,且同意这种变更,需要进一步确定由谁来支付变更所需要的费用。如果是用户要求的变更,则用户应支付这笔费用,否则必须认真进行成本/效益分析,以确定是否值得做这种变更。

(3) 变更批准或拒绝。配置控制委员会根据评估结果,决策是否可以变更。一般有四种结果:

① 批准变更。

② 拒绝变更。

③ 部分变更。需要指出应该变更的部分。

④ 延迟变更或待定。当前不具备变更条件或不能决定是否变更,等条件成熟时再决策。

无论哪种变更决策结果,都要通知变更请求人,并且保存所有相关记录。如果以后的事件证明拒绝变更是错误的,或者待定的变更,这些保存的记录是有用的。

(4) 变更实现。变更实现人员根据配置控制委员会给予的权限,在项目经理和配置控制委员会的指导下,从受控库中取出基线的拷贝,实现被批准的变更,并对实现的变更进行验证。一旦配置控制委员会认为正确实现并验证了一个变更,就可以将更新的基线放入配置库中,更新该基线的版本标识。变更实现过程如图13.12所示。在变更实现过程中,变更实现人员要填写变更

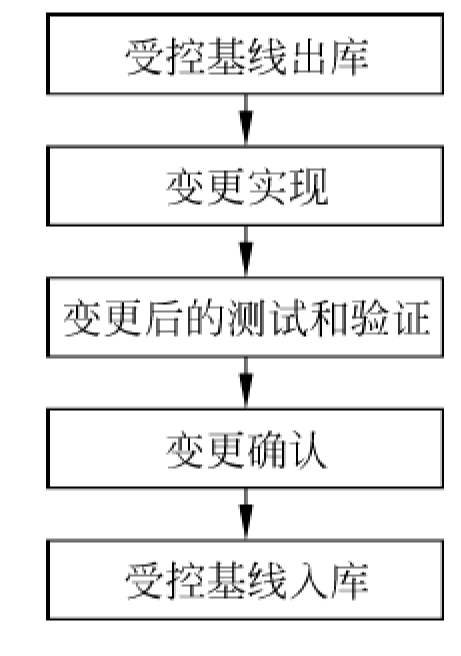


图13.12 变更实现过程

日志记录和变更测试记录。

13.4.4 状态报告

1. 状态报告

配置状态报告就是根据配置项操作数据库中的记录，向管理者报告软件开发活动的进展情况。配置状态报告记录了对已建立基线的各个项目的全部更改，能证实这些更改的形成过程是否符合更改控制的要求，是否符合非一致性报告及纠正措施过程要求。当进行验收时，配置状态报告信息将成为功能配置审核和物理配置审核的关键因素。

配置状态报告的任务是记录、报告整个生命周期中软件的状态，用以跟踪对已建立基线的需求、源代码、数据以及相关文档的更改、文件的形式等，表明每一软件版本的内容，以及形成该版本的所有更改，提供相关人员了解，以加强配置管理工作。

配置状态报告中要包括每一个配置项的状态报告，其格式如表 13.2 所示。

表 13.2 配置项状态报告

配置项名称		**配置项标识**	
文件名称		版本号	
经历的变更			
存放位置		维护人员	
配置管理员		报告日期	

2. 状态统计

配置状态统计是配置状态报告的组成部分之一，用于在产品开发过程中，基于已发现并修复了的缺陷类型、数量、频率和严重性等方面来说明产品的状态。状态报告持续地记录了配置状态以及保持基线产品基线变更的历史，并使相关人员了解配置和基线的状态。配置状态统计包括在软件生命周期中对基线所有变更的可跟踪性报告。

项目和配置项的关键信息可以通过配置状态统计传递给项目组成员。软件工程师可以看到做了哪些修改，或者每个文件包含在哪个基线中；项目经理可以跟踪项目的问题报告和各种其他维护活动。这样的报告应该定期进行，并尽量通过工具自动生成，用数据库中的实际数据来真实反映各配置项的情况。

3. 状态报告的主要内容

配置状态报告应根据报告着重反映当前基线配置项的状态，以作为对开发进度报告的参照；同时也能根据开发人员对配置项的操作记录，对开发团队的工作做进一步分析。配置状态报告主要包括以下内容：

(1) 基线和发布标识符。

(2) 为构建系统而使用的软件的最新版本。

(3) 对系统进行的变更次数。

(4) 基线和发布版本的数量。

(5) 配置项的使用和变动情况。

(6) 对基线和发布版本的比较结果。

13.4.5 配置审核

配置审核是根据需求规格说明或软件合同检验软件产品配置,验证每个软件配置项的正确性、一致性、完整性、有效性和可追踪性,以判定系统是否满足需求。

配置审核的任务是验证配置项对配置标识的符合性,其目的是证实软件生命周期中各项产品在技术和管理上的完整性,同时还要确保所有文档的内容变动不超出当初确定的软件需求范围,使得软件配置具有良好的可跟踪性。配置审核的对象主要是软件配置项的变化信息,包括软件配置项的创建时间、创建者、修改时间、修改内容、修改者等。

软件配置审核一般使用两种方法:正式技术审核和软件配置审核。正式技术审核在软件交付用户前实施,着重检查已完成修改的软件配置对象的技术正确性,评审者评价软件配置项,决定它与其他软件配置项的一致性,一般对所有的变更都要进行正式技术审核。软件配置审核作为正式技术审核的补充,评价在审核期间通常没有被考虑的软件配置项的特性。

1. 审核时机

配置管理人员对配置标识和配置数据库进行审核,确保配置数据库中的配置信息能真实反映软件基础架构中配置标识的存在和变更情况。审核的时机如下:

(1) 实施新的配置管理数据库(CMDB)后。

(2) 对软件基础架构实施重大变更前后。

(3) 在一项软件发布和安装被导入实际环境之前。

(4) 灾难恢复后或事故恢复正常后。

(5) 发现未经授权的 CI 后。

(6) 任何其他必须的时候。

2. 审核内容

配置审核包括两方面内容:配置活动审核与基线审核。配置活动审核用于确保项目组成员的所有配置活动都遵循已批准的软件配置管理标准和规范;基线审核用于保证基线化软件工作产品的完整性和一致性,且满足功能需要。

(1) 配置活动审核的审核项包括是否及时升级工作产品,是否执行配置库定期备份,是否定期执行配置管理系统病毒检查,评估配置管理系统是否满足实际需要,上次审核中发现的问题是否已全部解决等。

(2) 基线审核的审核项包括版本号、一致性(需求与设计、设计与代码的一致关系)、完整性(所有配置项是否纳入基线库,所有源文件是否存在于基线库,源文件是否可生成最终产品)。

在实际操作中,一般认为审核是一种事后行为,很容易被忽视。但“事后”是相对的,在软件项目初期发现的问题对项目后期工作具有重要的指导价值。为了提高审核效果,要进行审核跟踪。在软件项目进行过程中要定期进行配置审核,定期备份,保证备份介质的安全性和可用性。

思考题

1. 如何理解配置管理的定义?
2. 缺乏对软件开发过程的统一管理会产生哪些问题?
3. 配置管理的作用是什么?
4. 什么是软件配置项?软件配置项包括哪些状态?软件配置项如何分类?
5. 如何理解 IEEE 对基线的定义?为什么要建立基线?
6. 版本演变有哪两种方式?
7. 配置管理组织结构的构成及各成分的作用是什么?
8. 配置管理的主要活动包括哪些?
9. 如何进行版本的检入检出控制?
10. 怎样理解变更的波及面?
11. 如何进行变更控制?
12. 状态报告的主要内容包括哪些?
13. 如何把握配置审核的时机?
14. 配置审核的主要内容是什么?

第14章 文档管理

纵观软件开发历程，经常由于人员变动或其他原因，缺少相应的软件文档，使管理人员或新的开发人员查不到相关资料而无法继续工作，致使软件开发被迫终止，不得已进行二次开发，造成极大浪费；一些软件成果在维护阶段也因文档资料不健全，即以一种拒绝修改或无法修改的方式出现，以致软件在使用过程出现的错误无法更正，既不能完成对适应新环境的软件修改，又不能增强系统功能的开发，缩短了软件的使用寿命。因此，软件文档是软件项目开发成功的基础，软件开发中的文档管理势在必行。

14.1 文档管理概述

14.1.1 文档管理的概念

软件文档是与软件研制、维护和使用有关的材料，是以人们可读的形式表示的技术数据和信息。软件文档规定软件设计细节，说明软件功能，描述软件使用方法。软件文档和计算机程序一起构成一个完成特定功能的软件。

软件项目文档是软件项目开发中的重要组成部分。文档对于项目开发成功和项目维护起着重要的保障和支持作用。大型软件项目都应该有大量与系统相关的文档。有时，一个中小型项目也可能有成千上万行的文档。

一般来说，文档数量多少、规模大小、结构复杂程度都与所开发软件项目的规模大小和复杂程度成正比。在软件项目中，有很大一部分开发成本都发生在文档的准备、编制过程中，因而，项目管理者一定要对与项目有关的文档有足够的重视。

与软件项目相关的项目文档一般都具有以下一些属性：

(1) 作为开发组成员之间交流沟通的媒体。

(2) 为维护工程师提供有关信息系统的资料和知识库。

(3) 为管理者提供项目计划、预算、开发进度等各方面的信息。

(4) 作为最终用户手册或管理员手册，使用户知道如何使用、维护与管理系统。

14.1.2 文档与软件规模

一般来说，文档数量与软件规模成正比。大规模软件需要编制的文档数量较多；小规模软件需要编制的文档数量较少。可根据实际情况，对 GB/T 8567 所列文档进行合并或

分解。

当所开发的软件系统非常大时，一种文档可以分成几卷编写。例如，项目开发计划可分为进度计划、质量保证计划、配置管理计划、用户培训计划、安装实施计划等；系统设计说明书可分为系统设计说明书、子系统设计说明书；程序设计说明书可分为程序设计说明书、接口设计说明书、版本说明书；操作手册可分为操作手册、安装实施过程；测试计划可分为测试计划、测试设计说明、测试规程、测试用例；测试分析报告可分为综合测试报告、验收测试报告；项目开发总结报告可分为项目开发总结报告、资源环境统计。

《GB/T 8567—1988 计算机软件产品开发文件编制指南》所列文档与软件规模的关系如图 14.1 所示。如果按《GB/T 8567—2006 计算机软件文档编制规范》编写文档，可参考图 14.1 的内容进行合并或分解。



图 14.1 文档与软件规模

14.2 文档的分类与作用

14.2.1 文档分类

软件文档从形式上来看，大致可分为两类：

(1) 开发过程中填写的各种图表，称为工作表格。

(2) 编制的技术资料或技术管理资料，称为文档或文件。

按照文档产生和使用的范围，软件文档大致可分为开发文档、产品文档、管理文档三类。

1. 开发文档

开发文档是描述软件开发过程(包括软件需求、软件设计、软件测试、软件质量保证等)的一类文档，开发文档也包括软件的详细技术描述(如程序逻辑、程序间相互关系、数据格式和存储等)。这类文档的例子有软件需求规格说明书、数据要求说明书、概要设计说明书、详

细设计说明书、可行性研究报告、项目开发计划等。其主要作用如下：

（1）开发文档是软件开发过程中各个阶段之间的通信工具，记录软件需求、设计、编码和测试的详细规定和说明。

（2）开发文档描述开发小组的职责。通过规定软件、主题事项、文档编制、质量保证人员以及包含在开发过程中任何其他事项的角色来定义做什么、如何做以及何时做。

（3）开发文档用做检验点而允许管理者评定开发进度。如果开发文档丢失、不完整或过时，管理者将失去跟踪和控制软件项目的一个重要工具。

（4）开发文档形成了维护人员所要求的基本软件支持文档。而这些支持文档可作为产品文档的一个部分。

（5）开发文档记录软件开发历史。

2. 产品文档

产品文档是描述开发过程的产物，规定关于软件产品的使用、维护、增强、转换和传输等信息，供用户、运行者和维护人员使用。产品文档的例子有培训手册、参考手册、用户指南、软件支持手册、产品手册和信息广告等。

（1）产品文档的主要作用如下：

① 向使用和运行软件产品的任何人提供培训和参考信息。

② 便于未参加开发本软件的程序员进行维护。

③ 促进软件产品的市场流通或提高可接受性。

（2）产品文档用于下列类型读者：

① 用户。利用软件输入数据、检索信息和解决问题。

② 运行者。在计算机系统上运行软件。

③ 维护人员。维护、增强或变更软件。

（3）产品文档包括如下内容：

① 用于管理者的指南和资料。监督软件使用。

② 宣传资料。通告软件产品的可用性并详细说明它的功能、运行环境等。

③ 一般信息。对任何有兴趣的人描述软件产品。

3. 管理文档

管理文档从管理角度记录项目管理信息，这些信息包括开发过程每个阶段的进度和进度变更记录、软件变更情况记录、相对于开发的判定记录及职责定义。管理文档的例子有项目开发计划、测试计划、测试报告、开发进度月报、项目开发总结等。

14.2.2　文档作用

软件文档的作用主要体现在以下六个方面。

1. 管理依据

在软件开发过程中，管理者必须了解开发进度、存在的问题和预期目标。每一阶段计划安排的定期报告提供了项目的可见性。定期报告还提醒各级管理者注意该部门对项目承担的

责任以及该部门效率的重要性。开发文档规定若干个检查点和进度表，使管理者可以评定项目进度，如果开发文档有遗漏、不完善、内容陈旧，管理者将失去跟踪和控制项目的重要依据。

2. 任务之间联系的凭证

软件开发项目通常被划分成若干个任务，并由不同的小组完成。学科方面的专家建议立项，分析员阐述系统需求，设计员为程序员制定总体设计，程序员编制详细的程序代码，质量保证专家和审查员共同评价整个系统性能和功能的完整性，负责维护的程序员改进各种操作或增强某些功能。

这些人员需要的互相联系是通过文档资料的复制、分发和引用而实现的，因而，任务之间的联系是文档的一个重要功能。大多数系统开发方法为任务的联系规定了一些正式文档。例如，分析员向设计员提供正式需求规格说明，设计员向程序员提供正式设计规格说明。

3. 质量保证

负责软件质量保证和评估系统性能的人员需要程序规格说明、测试和评估计划、测试系统用的各种质量标准，以及关于期望系统完成什么功能和系统怎样实现这些功能的清晰说明；必须制定测试计划和测试规程，并报告测试结果；还必须说明和评估安全、控制、计算、检验例行程序及其他控制技术。这些文档的提供可满足质量保证人员和审查人员上述工作的需要。

4. 培训与参考

使系统管理员、操作员、用户、管理者和其他有关人员了解系统如何工作，以及为了达到各自的目的，如何使用系统。

5. 软件维护支持

维护人员需要软件系统的详细说明以帮助熟悉系统，找出并修正错误，改进系统以适应用户需求的变化或适应系统环境的变化。

6. 历史档案

软件文档是未来项目的一种资源。通常文档记载系统的开发历史，可使有关系统结构的基本思想被以后的项目利用。系统开发人员通过审阅以前的系统以查明哪些部分已试验过了，哪些部分运行很好，哪些部分因某种原因难以运行而被排除。良好的系统文档有助于把程序移植和转移到各种新的系统环境中。

14.3 文档编制要求

1. 基本要求

有关软件文档编制的基本要求如下：

（1）制定计划，标识在软件产品生命周期期间产生的文档。将计划形成文档，并加以实施。对每一种标识的文档，应阐述下述内容：标题或名称；目的；预期读者；有关输入、开

发、评审、修改、批准、生产、存储、发行、维护和配置管理的规程和职责；中间和最终版本的日程安排。

（2）每一种标识的文档应根据适用的文档编写标准进行编写，标准包括格式、内容描述、页码编号、插图表格安排、专利/保密标志、封装以及其他表达项目。

（3）确认文档输入数据的来源。可以使用自动化的文档编写工具。

（4）按照文档编写标准的格式、技术内容和表达方式要求，评审和编辑产生的文档。文档在发布之前应由适当的授权人员进行批准。

（5）按照计划产生和提供文档。文档的产生和发布可以使用纸张、电子媒体或其他媒体。应按照有关记录保存、保密安全性、维护和备份的要求存储主要资料。

（6）按照配置管理过程进行控制。

（7）在修改文档时需要执行的任务应按照维护过程进行。对于置于配置管理之下的文档，其修改按照配置管理过程进行管理。

2. 管理要求

管理者严格要求软件开发人员和编制组完成文档编制，并且在策略、标准、规程、资源分配和编制计划等方面给予支持。对项目管理者提出的有关软件文档管理的要求如下：

（1）管理者对文档工作的责任。管理者要认识到正式或非正式文档都是重要的，还要认识到文档工作必须包括文档计划、编写、修改、形成、分发和维护等方面。

（2）管理者对文档工作的支持。管理者应为编写文档的人员提供指导和实际鼓励，并使各种资源有效地用于文档编写。

（3）管理者的主要职责，有以下几项：

① 建立编制、登记、出版系统文档和软件文档的各种策略。

② 把文档计划作为整个开发工作的组成部分。

③ 建立确定文档质量、测试质量和评审质量的各种方法和规程。

④ 为文档各个方面确定和准备各种标准和指南。

⑤ 积极支持文档工作以便形成在开发工作中自觉编制文档的团队风气。

⑥ 不断检查已建立起来的过程，以便保证文档编制符合策略要求，并保证各种规程遵守有关的标准和指南。

（4）项目管理者在项目开发前应决定如下事项：

① 要求哪些类型的文档。

② 提供多少种文档。

③ 文档包含的内容。

④ 达到何种级别的质量水平。

⑤ 何时产生何种文档。

⑥ 如何保存、维护文档以及如何进行通信。

⑦ 如果一个软件合同是有效的，应要求文档满足所接受的标准，并规定所提供的文档类型、每种文档的质量水平以及评审和通过的规程。

14.4 文档编制过程

文档编制过程按图 14.2 所示的顺序进行。图中有两个阴影框,上一个阴影框中的所有活动应在下一个阴影框中的所有活动开始之前完成。在下面阴影框中的两个活动(评审文档、可用性测试)可以是并行的。虚线表示可能的重复。

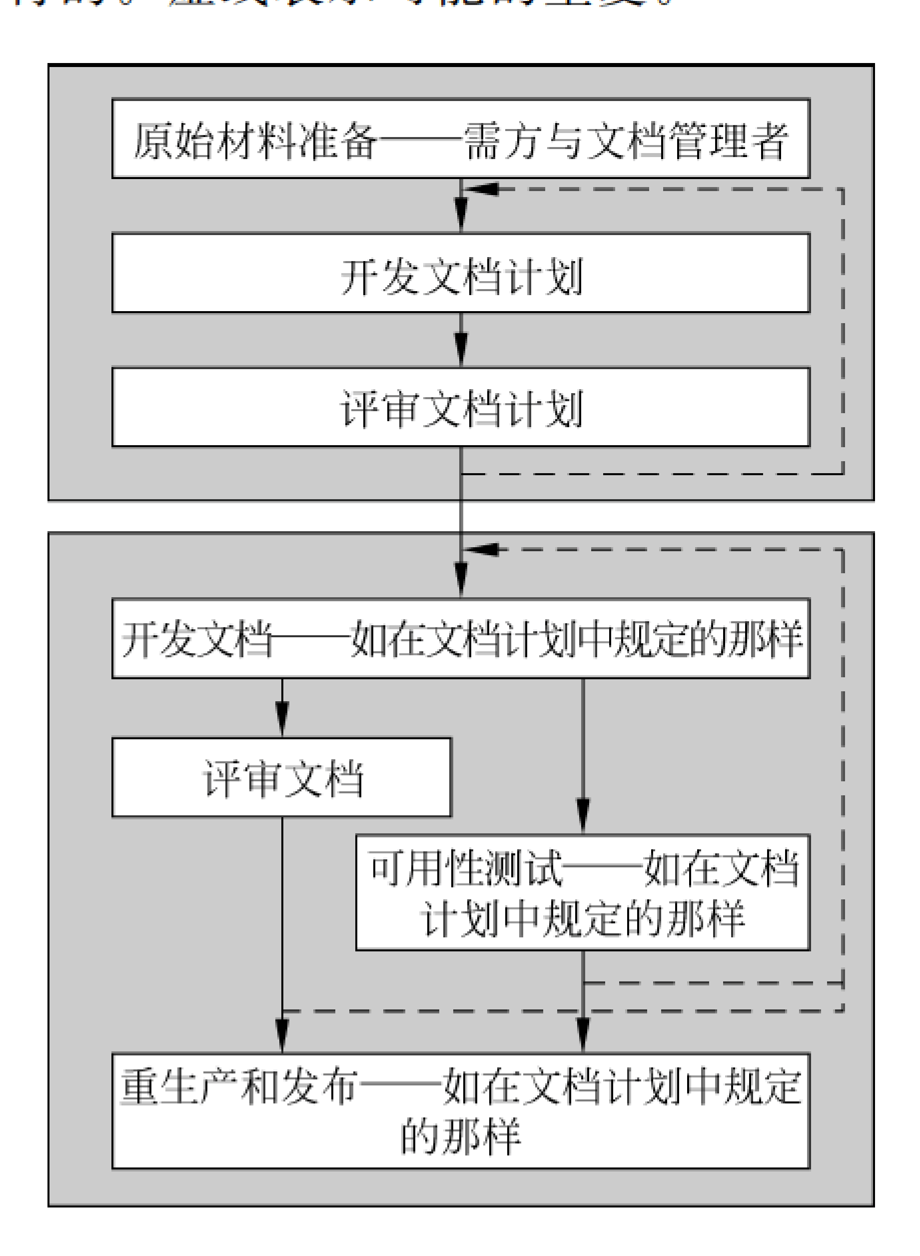


图 14.2 文档编制过程概要

开发文档活动应按照材料准备、文档计划、文档编制、文档编号、文档评审、文档签署、文档归档、文档保管和文档维护的逻辑顺序进行,如图 14.3 所示。

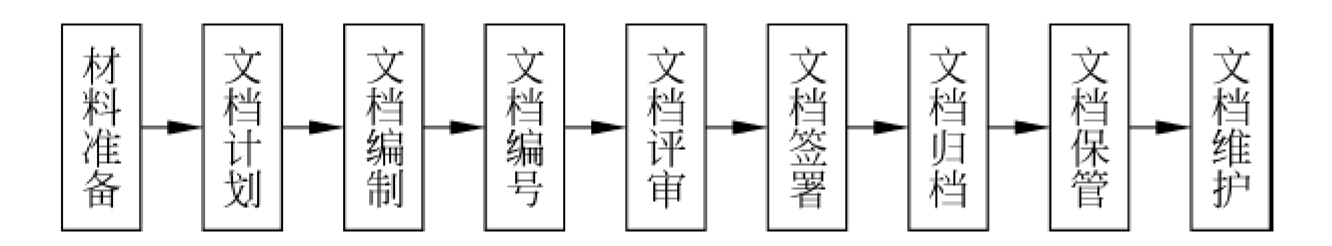


图 14.3 开发文档活动的逻辑顺序

由于文档编制涉及的内容较多,将在 14.5 节讲述,其他步骤在本节逐一讲述。

1. 材料准备

需方应允许文档管理者访问以下内容:

(1) 所有有关的规格说明、记录格式、屏幕和报告布局、CASE 工具输出以及文档准备所需要的任何其他信息。

(2) 若可用,软件的操作副本。

(3) 软件分析员和程序员,能及时和确切地解答由文档开发人员提出的问题。

(4) 若可能,访问典型用户(为了做读者分析和可用性测试)。

为了增加对产品和读者的了解，对软件开发人员的访问是必要的，但使这样的访问保持到最少是文档管理者的责任。

不管文档管理者是否是软件的开发者，需方应提供适用的标准、风格和格式指南以及其他相关材料。文档管理者应分发这些材料到需要它的文档开发人员。

保证需方交付给文档管理者的所有材料当交付时是完整的和正确的，且在交付后保持是最新的，这是需方的责任。

需方保证，提供的材料没有一个违反任何其他部门的知识产权。

文档管理者应采取相应步骤，以保证由需方提供的材料保持在很好的状态；应保证需方要求的信息安全，并在文档项目完成后，所有材料返回给需方。

2. 文档计划

对于具体的应用软件项目，项目负责人应根据实施规定，确定文档编制计划。其中包括编制哪几种文档，详细程度如何；各文档的编制负责人和进度要求；审查/批准负责人和时间进度安排；在开发时期内各文档的维护、修改和管理的负责人，以及批准手续。有关的开发人员必须严格执行文档编制计划。

编制计划工作应尽早开始，对计划评审应贯穿项目的全过程。如同任何其他计划一样，文档计划指出未来的各项活动，当需要修改时必须加以修改。导致对计划作适当修改的常规评审应作为项目工作的一部分，所有与计划有关的人员都应得到文档计划。

文档计划一般包括以下几个方面内容：

(1) 列出应编制文档的目录。

(2) 提示编制文档参考的标准。

(3) 指定文档管理员。

(4) 提供编制文档所需要的条件，落实文档编写人员、所需经费以及编制工具等。

(5) 明确保证文档质量的方法，为了确保文档内容的正确性、合理性，应采取一定的措施，如评审、鉴定等。

(6) 绘制进度表，以图表形式列出在软件生命周期各阶段应产生的文档、编制人员、编制日期、完成日期、评审日期等。

(7)明确文档计划及文档的分发情况，明确参与文档工作的所有人员职责。

此外，文档计划规定每个文档要达到的质量等级，以及为了达到期望的结果必须考虑哪些外部因素。

3. 文档编号

在文档编号阶段，应按各单位规定的编号方法对软件文档进行编号，以便于管理。编号方法有十进分类法、隶属法等，各单位可根据本单位的实际情况确定一种编号方法。不论采用何种方法，应使编号具有唯一性。例如，航天型号软件文档编号通常由单位代号(XX)、十进分类特征标记(Y. YYY)、登记顺序号(ZZZ)、文档简号(WW)四部分组成，源程序和执行程序文档应在文档简号后加媒体代号(M)，媒体代号与文档简号之间用横线隔开。结构如下：

XX Y. YYY ZZZ WW-M

(1) 单位代号(XX)：某单位的编号，通常由两个字符组成，例如 De。

(2) 十进分类特征标记(Y.YYY):按产品的技术特征给出的标记。例如,2.467 表示数据库管理系统。

(3) 登记顺序号(ZZZ):用于区分十进分类特征标记相同的不同产品的软件文档。登记顺序号由各单位标准化部门统一给定。通常由三位十进制数字组成。例如 021。

(4) 文档简号(WW):表示文档的名称,通常用两个字符组成,可按表 14.1 的英文缩写表示简号。

(5) 媒体代号(M):为区分源程序驻留的不同载体而给定的代号,用一位阿拉伯数字表示,如表 14.2 所示。

表 14.1 文档名称及文档简号

文档名称	文档简号	文档名称	文档简号
软件任务书	RW	组装测试计划	ZJ
可行性研究报告	RY	组装测试分析报告	ZF
项目开发计划	XJ	确认测试计划	QJ
软件需求规格说明	RX	确认测试分析报告	QF
概要设计说明	GS	系统联试计划	XJ
详细设计说明	XS	系统联试分析报告	XF
用户手册	YC	源程序	CX
单元测试计划	DJ	可执行程序	KX
单元测试分析报告	DF		

表 14.2 媒本及其代号

媒体代号	媒体名称	媒体代号	媒体名称
2	磁带	6	只读存储器
4	硬磁盘	8	光盘
5	软磁盘	9	其他媒体

4. 文档评审

文档评审十分重要,文档评审必须与技术评审相结合。为了提高软件产品质量,在软件开发每个阶段,应对该阶段形成的文档进行严格评审,以便尽早发现问题并及时采取措施加以解决,从而确保文档内容的正确性,避免或减少返工,并为进入下一阶段工作做好组织上和技术上的准备。

对一些大的项目,正规评审通常在开发方法学指导下进行。正规评审应包括文档评审,以保证文档不但正确,而且内容是最新的。

评审通常采用评审会的方式进行,步骤为:

(1) 由软件开发单位负责人、用户代表、开发小组成员、科技管理人员和标准化人员等组成评审小组,必要时还可邀请外单位的专家参加。

(2) 开会前,由开发单位负责人确定评审的具体内容,并将评审材料发给评审小组成员,做好评审准备。

(3) 由开发单位负责人主持评审会,根据文档编制者对文档的说明和评审条目,由评审

小组成员进行评议、评审，评审结束应给出评审结论，评审小组成员应在评审结论上签字。

根据评审对象，软件评审可分为需求评审、设计评审和其他评审。无论项目大小或项目管理的正规化程度如何，需求评审和设计评审都是必不可少的。需求必须清楚，用户和开发者双方都必须理解需求，为了将需求转换成程序及程序成分，设计的细节必须经过同意并写成文档。因此，应特别重视需求评审和设计评审工作。

需求评审进一步确认开发者和设计者已了解用户要求什么，以及用户从开发者一方了解某些限制和约束。需求评审（可能需要一次以上）产生一个被认可的需求规格说明。基于对系统要做什么的共同理解，才能着手进行软件设计。用户代表必须积极参与开发和需求评审，参与对需求文档的认可。

设计评审通常包括概要设计评审和详细设计评审。在概要设计评审过程中，主要详细评审软件概要设计规格说明和软件组装测试计划，概要设计规格说明应根据概要设计评审结果加以修改。详细设计评审主要评审详细设计规格说明和软件单元测试计划。

如果存在已有的或制定的标准或指南，则可对照这些标准或指南来评审文档。正规评审要保证产品文档是准确的、完整的，而且是适合读者的。

5. 文档签署

在文档签署阶段，应按有关规定对所有文档进行签署。文档签署一般按图 14.4 所示的顺序进行。其中，会签仅在必要时才进行。签署不得代签。修改单的签署与被修改的文档签署相同。

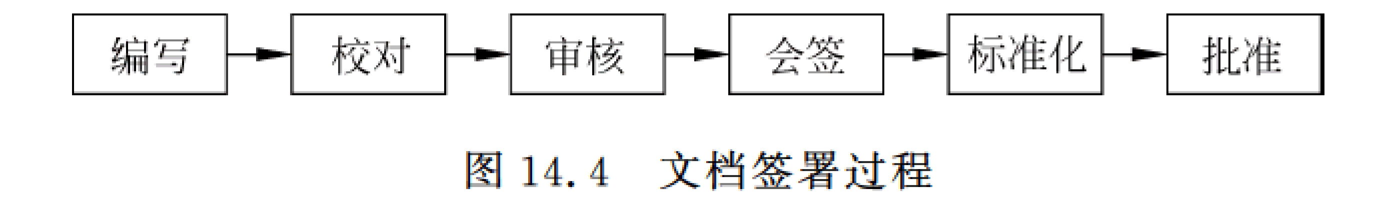


图 14.4 文档签署过程

各签署者的技术责任如下：

（1）“编写”签署者应对所编写软件文档的正确性、合理性、继承性、可靠性、完整性、一致性负责。

（2）“校对”签署者应对所校对的软件文档与“编写”签署者负同等责任。

（3）“审核”签署者应对下述内容负责：设计方案选择的正确性、合理性，是否贯彻总体设计意图，设计结果或结论是否正确、全面，图例、术语、符号、公式、单位等是否贯彻标准和有关规定。

（4）“会签”签署者应对所会签的软件文档有关部分的正确性、合理性、完整性、协调性负责。

（5）“标准化”签署者应对下述内容负责：编写者是否贯彻先行标准和有关规章制度，软件文档的完整性和签署是否完整正确。

（6）“批准”签署者应对下述内容负责：功能、性能指标是否达到规定的技术要求；软件文档的正确性、合理性和协调性；软件文档的完整性及设计质量；软件文档是否贯彻现行标准及有关规章制度。

6. 文档归档

在文档归档阶段，应将通过评审和签署的软件文档及时归档。归档的文档应包括整个

软件生命周期内形成的全部文档。文档归档应满足如下条件：

(1) 归档的文档应是经过鉴定或评审的。

(2) 文档应签署完整、成套、格式统一、字迹工整。

(3) 印制本、打印本以及各种报告应装订成册，并按规定进行编号和签署。

软件文档应在软件开发过程每个阶段结束后及时归档。

7. 文档保管

在文档保管阶段，应对已归档的软件文档进行妥善保管。对一个单位来说，软件文档是极其重要的财富，代表一种在时间、思想和能力上的具有特殊意义的投资。此外，软件文档描述了在另一方面(开发工作和产品)的重要投资。通常应在不同地点建立设施来保存重要文档副本。这种非现场保存应存储所有开发文档和产品文档的后备副本。如果文档是联机开发的，则应将它存储到磁带或磁盘上，以便能够迅速将这些文档转换成可用的形式。在人为事故或自然灾害情况下，可以使用备用的磁带、磁盘、软盘、清单、系统图等重新构造系统。文档保管包括登记、保存、借阅和修改四项工作。文档保管应按国家和上级主管部门的要求，以及各单位的规章制度规定执行。

8. 文档维护

在文档维护阶段，当出现程序错误、文档错误或适应新的环境和需求时，应对软件产品和文档进行维护。维护过程大致按如下步骤进行：

(1) 软件产品完成后，确定软件维护人员。

(2) 在软件运行过程中，由软件维护人员负责收集记录从用户或其他途径反馈回来的软件质量信息。

(3) 在分析整理后，按照 GB/T 8567 的要求填写“问题报告单”。

(4) 由软件维护人员或原开发人员根据“问题报告单”中提出的要求，填写“修改申请报告”。

(5) 根据申请批准的“修改申请报告”，由软件维护人员或原软件开发人员对程序进行修改和测试。

(6) 在测试通过后，由软件维护人员或原软件开发人员填写“软件修改报告”，并修改相应的文档和媒体。

(7) 将“软件修改报告”和修改后的文档送校对、审核、会签、标准化和批准等有关负责人审阅和签署。

(8) 将修改后的文档、媒体和“软件修改报告”送交档案管理部门存档。

(9) 软件维护人员汇总并整理修改的情况，在此基础上编写“软件维护通报”。

14.5 文档编制

14.5.1 编制策略

软件文档的编制可以用自然语言，特别设计的形式语言，介于二者之间的半形式语言(结构化语言)，各类图形、表格等来编制文档。文档可以书写，也可以在计算机支持系统中

产生,但必须可阅读。

文档策略是由上级(资深)管理者准备和支持的,对下级开发单位或开发人员提供指导,并不是做什么或如何做的详细说明。一般说来,文档编制策略陈述要明确,通告到每个人并且理解它,进而使策略被有效地贯彻实施。文档编制策略如下:

(1) 文档需要覆盖整个软件生命周期。在项目早期几个阶段就要求有文档,而且在贯穿软件开发过程中必须是可用的、可维护的。在开发完成后,文档应满足软件的使用、维护、增强、转换和传输。

(2) 文档应是可管理的。为指导和控制文档的获得和维护,管理者和发行专家应准备文档产品、进度、可靠性、资源、质量保证和评审规程的详细计划大纲。

(3) 文档应适合于其读者。读者可能是管理者、分析员、无计算机经验的专业人员、维护人员、文书人员等。根据任务的不同,要求不同的材料表示和不同的详细程度。针对不同的读者,发行专家应负责设计不同类型的文档。

(4) 文档作用应贯穿软件整个开发过程。在软件整个开发过程中,应充分体现文档的作用和限制,即文档应指导全部开发过程。

(5) 文档标准应被标识和使用。应尽可能地采纳现行标准,若没有合适的现行标准,必要时应需研制适用的标准或指南。

(6) 应规定支持工具。工具有助于开发和维护软件产品,包括文档。因此尽可能地使用经济且可行的工具。

14.5.2 质量等级

仅仅依据规章、传统的做法或合同的要求来制作文档是不够的。管理者还必须确定文档的质量要求以及如何达到和保证质量要求。

质量要求的确定取决于可得到的资源、项目的大小和风险,可以对产品每个文档的格式及详细程度做出明确规定。

每个文档的质量必须在文档计划期间就有明确规定。在《GB/T 16680—1996 软件文档管理指南》中,对文档质量按文档的形式和列出的要求划分为四级。

(1) 最低限度文档(1 级文档):1 级文档适合开发工作量低于一个人月的开发者自用程序。该文档应包含程序清单、开发记录、测试数据和程序简介。

(2) 内部文档(2 级文档):2 级文档可用于在精心研究后被认为似乎没有与其他用户共享资源的专用程序。除 1 级文档提供的信息外,2 级文档还包括程序清单在内的足够的注释以帮助用户安装和使用程序。

(3) 工作文档(3 级文档):3 级文档适合于由同一单位内若干人联合开发的程序,或可被其他单位使用的程序。

(4) 正式文档(4 级文档):4 级文档适合要正式发行可供普遍使用的软件产品。关键性程序或具有重复管理应用性质(如工资计算)的程序需要 4 级文档。4 级文档遵守 GB 8567 的有关规定。

质量方面需要考虑的问题既要包含文档结构,也要包含文档内容。文档内容可以根据正确性、完整性和明确性来判断。而文档结构由各个组成部分的顺序和总体安排的简单性来测定。要达到这四个质量等级,需要投入的资源逐级增加,质量保证机构必须处于适当的

行政地位以保证达到期望的质量等级。

14.5.3 质量要求

如果不重视文档编写工作，或对文档编写工作安排不当，就不可能得到高质量文档。质量差的文档使读者难于理解，给使用者造成许多不便，而且削弱对软件的管理（难以确认和评价开发工作的进展情况），提高软件成本（一些工作可能被迫返工），造成误操作等。对编制高质量文档的要求如下。

1. 针对性

文档编制前应分清读者对象，每一种文档都有其特定的读者。这些读者可能包括软件开发小组成员、开发单位人员、用户、管理人员等。每种文档的编写必须适应读者的特点、水平和要求，力求做到易读易懂。文档为适应不同层次的读者，需要编制不同的类型。管理文档主要面向管理人员，用户文档主要面向用户，这两类文档不应像开发文档（面向开发人员）那样过多使用软件专用术语。

2. 精确性

文档的行文应当十分确切，不能出现多义性描述。同一课题几个文档的内容应当是协调一致、没有矛盾的。文档编写应力求简明，如有可能，配以适当的图表，以增强其清晰性。

3. 及时性

文档的编制应尽量做到随着开发工作的逐步深入而随时进行，并将每一个开发步骤内做出的决定和取得的工作成果及时形成文档。

4. 完整性

任何一个文档都应当是完整、独立、自成体系。例如，前言部分应做一般性介绍，正文给出中心内容，必要时还有附录，列出参考资料等。同一软件项目的几个文档之间可能有些部分内容相同，这种重复是必要的。不要在文档中出现转引其他文档内容的情况。例如，不要出现一些段落没有具体描述，用“见××文档××节”的方式。

5. 重复性

标准中列出的文档编制规范的内容要求显然存在某些重复。较明显的重复有两类：第一类是引言，引言是每一种文档都要包含的内容，以便向读者提供总体梗概；第二类是各种文档中的说明部分，如对功能、性能的说明，对输入、输出的描述，系统中包含的设备等。这是为了方便每种文档各自的读者，每种文档应该自成体系，尽量避免读一种文档时又不得不去参考另一种文档的情形。当然，在每一种文档里，有关引言、说明等同其他文档相重复的部分，在行文、所用的术语、详细的程度上，还是应该有一些差别以适应各种文档不同读者的需要。

6. 灵活性

鉴于软件开发是具有创造性的脑力劳动，不同软件在规模上和复杂程度上差别极大，在

文档编制工作中允许有一定的灵活性。这种灵活性包括以下内容：

（1）应编制的文档种类。可根据具体情况取舍，当项目规模、复杂性和失败风险增大时，文档编制范围、管理手续和详细程度将随之增加，反之，则可适当减少。

（2）文档的详细程度。详细程度取决于任务规模、复杂性和项目负责人对软件的开发过程及运行环境所需要的详细程度的判断。

（3）文档扩展。当被开发系统的规模非常大时，一种文档可以分成几卷编写，可以按其中每一个系统分别编制，也可以按内容划分成多卷。

（4）章、条的扩张与缩并。在软件文档中，一般宜使用标准提供的章、条标题。但所有章、条都可以扩展，可以进一步细分，以适应实际需要。反之，如果章、条中的有些细节并非必需，也可以根据实际情况缩并，此时章、条编号应相应地变更。

（5）程序设计的表现形式。标准对于程序设计的表现形式并未做出规定或限制，可以使用流程图的形式、判定表的形式，也可以使用其他表现形式，如程序设计语言、问题分析图等。

（6）文档的表现形式。标准对于文档的表现形式未做出规定或限制。可以使用自然语言，也可以使用形式化语言，还可以使用各种图、表。

（7）文档的其他种类。标准中规定的文档种类尚不能满足某些应用部门的特殊需要时，可以建立一些特殊的文档种类要求，例如软件质量保证计划、软件配置管理计划等，这些要求可以包含在本单位的文件编制实施规定中。

7. 可追溯性

由于在各个开发阶段编制的文档与各阶段完成的工作有着密切关系，前后两个阶段生成的文档，随着开发工作的逐步扩展，具有一定的继承关系。在一个项目各开发阶段之间提供的文档必定存在可追溯关系。例如，对于需求规格说明中的某一需求，必定在数据要求说明、软件设计说明、软件测试说明中有所体现，必要时能做到跟踪追查。

为使软件文档能起到以下作用：多种桥梁作用、有助于程序员编制程序、有助于管理人员监督和管理软件开发、有助于用户了解软件工作和应做的操作、有助于维护人员进行有效的修改和扩充，对文档的编制必须保证质量。软件管理者应严格要求软件开发人员和文档编制组完成文档编制，并且在策略、标准、规程、资源分配和编制计划等方面给予支持。

14.5.4　书写风格

如果希望产生好的文档，质量标准和质量评价是至关重要的，但是文档质量主要依赖于编写者文字的组织能力。简单地说，好的文档需要好的文笔。

编写文档不是件容易的事，同时也不是单纯的项目阶段过程。文档必须经过写、读、审阅和修改等过程，直到产生一个满意的文档。技术文档不是一种科学而是一种技巧，编写文档时，应注意以下问题：

（1）文档层次。根据文档的内容需要，安排标题层次，要求正文各项标题层次必须分明，各级标题包含相应内容，文档前后标题层次要取得平衡和统一。根据《GB 1.1—87 标准化工作导则标准编写的基本规定》，标题层次一律用阿拉伯数字连续编号，不同层次的数字之间用下圆点相隔（即圆点加在数字的右下角处），最末数字后面不用圆点。文档层次的划

分一般不超过四节，四节不够时，可将层次再细划分。

(2) 注释说明汇集表。符号、标志、缩略词、首字母缩写、计量单位、名词、术语等的注释表，及注释说明汇集表，应置于图表清单之后。

(3) 编号规则。文档中的图、表、附注、参考文献、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。序号可以全篇文档统一按出现先后顺序编码，对长篇文档也可以分章依序编码。标注形式应便于互相区别，可以分别为图 1、图 2.1；表 2、表 3.2；附注 C；文献[4]；式(5)、式(3.5)等。

(4) 页码。一律用阿拉伯数字连续编页码。页码由书写、打字或印刷的首页开始，作为第 1 页。封面、封二、封三、封底等都不编入页码。可以将题名页、序、目次页等前置部分单独编排页码。页码必须标注在每页的相同位置，便于识别。力求不出空白页，如有空白页，仍以有内容页作为单页页码。在一个总题下装成两册以上时，应连续编页码。如各册有副题名，则可分别独立编页码。

(5) 图。图包括曲线图、构造图、示意图、图解、框图、流程图、记录图、布置图、地图、照片、图版等。图应具有"自明性"，即只看图、图题和图例，不阅读正文，就可理解图意。图应编排序号。每一图应有简短确切的题名，连同图号置于图下。必要时，应将图上的符号、标记、代码等，用最简练的文字横排于图题下方，作为图例说明。曲线图的纵横坐标必须标注"量、标准规定符号、单位"，此三者只有在不必要标明(如无量纲等)的情况下方可省略。坐标上标注的量的符号和缩略词必须与正文中一致。照片要求主题和主要显示部分轮廓鲜明，便于制版。如用放大或缩小的复制品，必须清晰，反差适中。照片上应该有表示目的物尺寸的标度。

(6) 表。表的编排，一般是内容和测试项目由左至右横读，数据依序竖排。表应有自明性，表应编排序号。每一表应有简短确切的题名，连同表号置于表上。必要时应将表中的符号、标记、代码以及需要说明事项，以最简练的文字横排于表题下，作为表注，也可以附注于表下。表的各栏均应标明"量或测试项目、标准规定符号、单位"。只有在无必要标注的情况下方可省略。表中的缩略语和符号必须与正文中一致。表内同一栏的数字必须上下对齐。表内不宜用"同上"、"同左"等类似词，一律填入具体数字或文字。

(7) 符号和缩略词。符号和缩略词应遵照国家标准的有关规定执行。如无标准可循，可采纳本专业权威性机构或学术团体所公布的规定。也可以采用全国自然科学名词审定委员会编印的各学科词汇用词。如不得不引用某些不是公知公用的，且又不易为同行读者所理解的，或系作者自定的符号、记号、缩略词、首字母缩写字等时，均应在第一次出现时加以说明，给以明确的定义。

(8) 附录。附录作为文档的补充项目，并不是必需的。附录可编于后，也可以另编成册。依序用大写正体 A、B、C、……编序号。例如附录 A。附录中的图、表、式、参考文献等另行编序号，与正文分开，也一律用阿拉伯数字编码，但在数码前冠以附录序码，例如图 A1、表 B2、式(B3)、文献[A5]等。下列内容可考虑加入附录：为了整篇文档材料的完整，但编入正文又有损于编排的条理和逻辑性，这一类材料包括比正文更为详尽的信息、研究方法和技术更深入的叙述；由于篇幅过大或取材于复制品而不便于编入正文的材料；不便于编入正文的罕见珍贵资料；对一般读者并非必要阅读，但对本专业同行有参考价值的资料；某些重要的原始数据、数学推导、计算程序、框图、结构图、注释、统计表、计算机打印输出件等。

附录与正文连续编页码。每一附录均另页起。

(9) 段落简短。作为一个通用的规则,一个段落中包含的句子不易过多,特别是有转折含义时,应当尽量放在不同的段落中。通常人获得即时信息的能力是有限的,在短的段落中,概念能比较容易地维持在短期记忆里。

(10) 在写作时还应注意:运用主动而不是被动语气;运用正确的术语和语法结构,太多别字、拼写错误和语法错误会导致读者对文档丧失信心;句子不易过长,尽量用短句,使读者不需要把几句连贯起来才能理解其中要表达的意思;用词简洁,不要长篇累牍,质量比数量更重要;尽可能地逐条列举事实,合适的举例对于事实的理解比单纯的语句解释更清晰;不要用第一人称和第二人称,一律用第三人称。文档应该要像程序一样,采用同样的方法进行检测。在文档检测期间,应该用带有批评的眼光来挑出文档缺点,提出建议,从而达到提高文档质量的目的。在后期,主要侧重在错误的发现上而不再是错误的修改机制了。

思考题

1. 理解文档管理的概念。
2. 文档与软件规模有怎样的关系?
3. 按照文档产生和使用的范围,软件文档可分为哪几类?
4. 开发文档的主要作用是什么?
5. 软件文档编制的基本要求是什么?
6. 管理者对文档编制的主要职责有哪些?
7. 简述文档的编制过程。
8. 文档计划一般包括哪几方面内容?
9. 各单位应如何制定文档编号的规则?
10. 简述文档各签署者的技术责任。
11. 如何进行文档维护?
12. 如何理解文档编制策略?
13. 如何划分文档的质量等级?
14. 编制高质量文档有哪些具体要求?
15. 编写文档时,在书写风格方面应注意哪些问题?

第15章 人力资源管理

影响软件生产率的因素很多,但对生产率冲击最大的,是软件开发的人员和团队素质。由于软件开发不需要使用大量的物质资源,而主要是人力资源,软件开发与人的相关性很大。随着软件产业的不断发展,人们逐渐认识到:“人是软件公司最重要的资产。”人的因素决定软件企业或者项目的成败。

很多软件项目经理认为有效地进行人力资源管理是软件开发面临的最艰巨挑战,一个软件项目要想获得成功必须进行有效的人力资源管理。人力资源管理是项目管理中至关重要的组成部分,要充分调动人员的积极性,最大限度地发挥每个参与人员的作用。对人员的配置、激励、调度贯穿整个软件过程。人员的组织管理是否得当,也是影响软件项目开发质量的决定性因素。

15.1 软件项目人力资源的特征

软件项目人力资源有着不同于其他行业的特征,主要表现在以下几个方面:

(1) 软件从业人员具有年轻化的特征。由于软件从业人员大多是直接来自高等院校的应届毕业生,或是毕业时间不久的年轻人,基本年龄在20~30岁之间,这是从业人员年轻化的客观基础。另一方面,软件企业是依靠员工的知识创新生存的,产品实际是员工的思想、智力和创造性的劳动,人才的最佳创造年龄为25~45岁,37岁为峰值年,一旦错过这个时期,人力资源就会贬值,工作能力和创造力就会下降。在竞争激烈的软件产业,也要求保持人力资源队伍的年轻化,以保持企业的创新能力。

(2) 软件从业人员总体具有较高的文化素质。软件企业是高新技术企业,要求员工具备较高的文化素质和专业技术能力。绝大部分从业人员是通过接受高等教育途径达到这种要求的。国内外的大部分高等院校和研究机构都开设了计算机方面的专业和课程。许多国家专门成立软件学院,来培养软件行业的专门人才。许多国际巨型软件企业也同政府、高校联合培养软件人才。据有关资料统计,我国软件企业中,大专层次的从业人员占20%左右,本科层次的从业人员占60%左右,硕士、博士层次的从业人员占20%左右。

(3) 软件从业人员具有学习能力强的特点。软件从业人员具有较强的获取知识、信息,以及加工处理、应用知识和信息的能力。软件人员普遍具有良好的教育背景,培养了良好的学习习惯,掌握了学习的经验和方法。另一方面,软件技术的飞速发展和企业不断创新的要求,使软件从业人员时刻感受到学习的压力;软件产品和工具的快速推陈出新,使得软件从

业人员拥有的知识很快被淘汰或过时。只有通过不断学习，才能保持工作能力和竞争能力。软件从业人员在工作中的创新不仅仅是一种劳动和职责，他们也在创新和劳动成果中获得实现自我价值的满足感和快乐。学习是创新的必由之路，所以软件从业人员对学习新软件、新方法、新工具有自身内在的动力。

(4) 软件从业人员具有较强的自主性。知识型员工的高素质、高智商和高知识水平，提高了主观能动性，因而在工作中具有较强的自主性。与流水线上的操作工人被动地适应设备运转相反，知识型员工更倾向于拥有自主的工作环境，不愿意受制于人，而愿意自我引导。这种自主性也表现在工作场所、工作时间的灵活性，宽松的组织氛围等。创造性的劳动必然要求工作方式和工作过程的自主性，限制和干预只会影响软件从业人员能力的发挥和灵感的产生。

(5) 软件从业人员对工作的期望值高、成就感强。与其他行业相比，知识型员工更在意自身价值实现，强烈期望得到社会认可，并不满足于被动地完成一般性事务，而是尽力追求完美的结果。因此，软件人员热衷于具有挑战性的工作，把攻克难关看做是一种乐趣，一种体现自我价值的方式。许多软件人员选择加入一家软件公司，是出于公司的技术先进性和业内专业方向上的成长性，而其他诸如薪水、福利等物质条件则放在次要位置。离开某家公司的理由大多是工作“没意思”，也就是公司的工作达不到软件人员的期望值。

(6) 软件人才缺口大，两级人才严重不足。计算机在各行各业的应用日益普及，各种各样的应用软件需求也越来越多，软件产业成为投资热点，从而引起对软件从业人员的大量需求，但是目前在整个社会中从事这个行业的人员比较少。在我国软件队伍中，70%是年轻的软件工程师，但高级技术人才(比如高级系统分析员、项目总体设计师)和实际操作人员比较缺乏，软件人才结构呈两头小、中间大的橄榄型，这是制约我国软件产业发展的一大障碍。我国软件人才结构如图 15.1 所示。

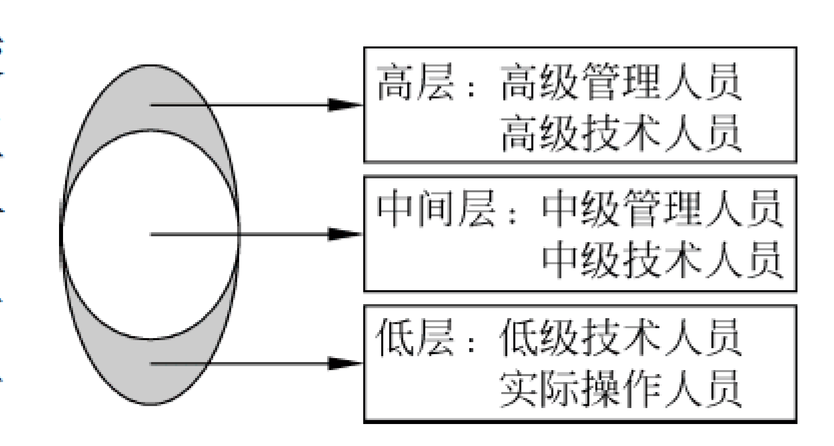


图 15.1　我国软件人才结构

(7)软件从业人员具有较高的流动性。知识经济对传统的雇佣关系提出了新的挑战，“资本雇佣劳动”这个定律开始受到质疑。知识管理专家玛汉·坦姆仆经过大量的实证研究认为知识型员工注重的前四个因素依次为个体成长、工作自主、业务成就和金钱财富，与成长、自主和成就相比，金钱的边际价值已退居相对次要地位。在所有行业中，软件从业人才流动频率最高。有数据显示，我国软件企业研发人员年流动率高达 40%，是社会平均流动率的四倍左右，且有些软件企业的人员流动率高达 50%～70%。智力资源的新陈代谢对软件行业来说本来是好事，能增加企业的开发能力，但频繁的流动使软件项目不能按时完成、核心技术泄露、大量培训费付之东流。流动方向基本上是国企流向外企，外企流向国外。

(8) 员工业绩难以量化考核。软件项目开发工作具有抽象性，项目成员的设计和编程工作是一种智力劳动，工作业绩有别于其他行业，业绩价值存在潜在性和评判标准的不确定性，因而工作业绩较难量化准确计量。比如，管理者无法根据编程实现 1 万行还是 7 千行代码来判断软件工程师的工作业绩，因为和 1 万行代码相比，也许 7 千行代码更有效率、更规范、更具有知识价值。

15.2 人力资源管理的主要内容

软件企业的人力资源管理由一系列相互联系的活动组成，包括人力资源需求预测与规划、人员的招募与甄选、培训与开发、绩效评价、薪酬管理以及建立和维护有效的员工关系等。

(1) 人力资源需求预测与规划。人力资源需求预测是指根据企业的发展规划和内外条件，选择适当的预测技术，对人力资源需求的数量、质量和结构进行预测。人力资源计划包括宏观计划和微观计划。宏观计划是指计划和预测组织短期和长期的人力资源需求；微观计划是指根据技能和能力的需求对组织的职务进行分析，即工作分析。人力资源计划在人力资源管理活动中的作用是：确定组织在目前以及未来人员需求的种类及数量；确定人员的招募方式，是通过外部招聘还是内部工作调动和晋升；确定组织人力资源开发和培训方面的需求。人力资源需求预测与规划是影响整个组织人员配置、培训和开发的主要因素。

(2) 人员的招募与甄选。人员招募是指组织根据人力资源规划和工作分析的要求，把具有一定技巧能力和其他特性的申请人吸引到企业或组织的空缺岗位上，以满足组织或企业人力资源需求的过程。而甄选就是利用各种工具和方法，为每个岗位挑选最佳职位候选人。目前，许多优秀的软件企业都通过职业素质测试、小组讨论、管理评价中心等，借助专业的招聘机构来挑选企业所需的人才。

(3) 培训与开发。培训与开发在软件企业的人力资源管理中具有独特地位。这是因为软件企业尤其要求员工具有快速学习、快速创新的能力，另外，软件企业员工本身具有个人价值实现、追求终身就业能力的强烈愿望。因此，对员工提供职业培训和职业生涯设计，进行全方位的人力资源开发，成为许多软件企业的共识。这类人力资源培训与开发包括职业技能培训、能力素质提升培训和管理培训等。

(4) 绩效评价。绩效评价就是根据员工个人的绩效标准来对其当前及过去的绩效进行评价。绩效评价的过程包括设定工作标准；根据标准来对员工的实际绩效进行评价；向员工提供反馈，以激励员工消除缺陷或者是继续保持优良的绩效。软件企业在绩效评价方面似乎比一些传统行业做得更好，一些企业正在寻求更好的办法来留住员工，提高员工绩效，解决绩效评价中出现的问题。通过绩效评价反馈系统来帮助绩效评价较差的员工提高绩效。另外，绩效评价结果成为软件企业确定员工培训需求和薪酬水平的依据。

(5) 薪酬管理。薪酬管理是指一个组织，针对所有员工提供的服务，来确定他们应当得到的报酬总额以及报酬结构和报酬形式的过程。在这个过程中，企业就薪酬水平、薪酬体系、薪酬结构、薪酬构成以及特殊员工群体的薪酬做出决策。同时，企业还要持续不断地制定薪酬计划，拟定薪酬预算，就薪酬问题与员工进行沟通，同时对薪酬系统的有效性做出评价后不断予以完善。薪酬管理的活动主要有管理直接薪酬、提高以绩效评价为基础的工资以及管理间接福利。其中，基于绩效的薪酬等级评定制度越来越成为软件企业薪酬管理的主要部分。

(6) 建立和维护有效的员工关系。改善工作环境是人力资源管理活动的一个重要方面。组织中人力资源竞争力的整体提高需要通过建立和维护有效的员工关系来实现。这些活动主要包括尊重员工权利；提供安全和健康的工作场所；在组织活动过程中，了解员工

使用的开发工具和工作方式；和员工及组织代表协商解决员工投诉。在这些活动中，一个主要的活动是改善工作环境，最大限度地保证员工的安全和健康，否则，势必造成成本上升甚至违反法律。

15.3　人员的组织与分工

大型软件项目需要很多人通力合作，花费一年甚至数年时间才能完成。为了提高工作效率、保证工作质量，软件项目人员组织、分工与管理是一项重要和复杂的工作，直接影响到软件项目的成败。构建软件开发团队取决于可供选择的人员、项目的需求以及组织的需求。

15.3.1　项目组的组织形式

软件人员之间联系的多少和方式与生产率直接相关。如果项目组内人数少，如2～3人，人员之间的联系比较简单。但在增加人员数目时，相互之间联系的复杂度增加，超过了线性关系的增长。因此，开发中的软件项目在任务紧张、延误进度的情况下，不提倡增加新的人员给予协助。除非分配给新成员的工作难度较小，不太费时即可上手，最好是分配比较独立的任务。

项目组内部人员的组织形式对生产率有重要影响，现有的组织形式有三种，如图15.2所示。

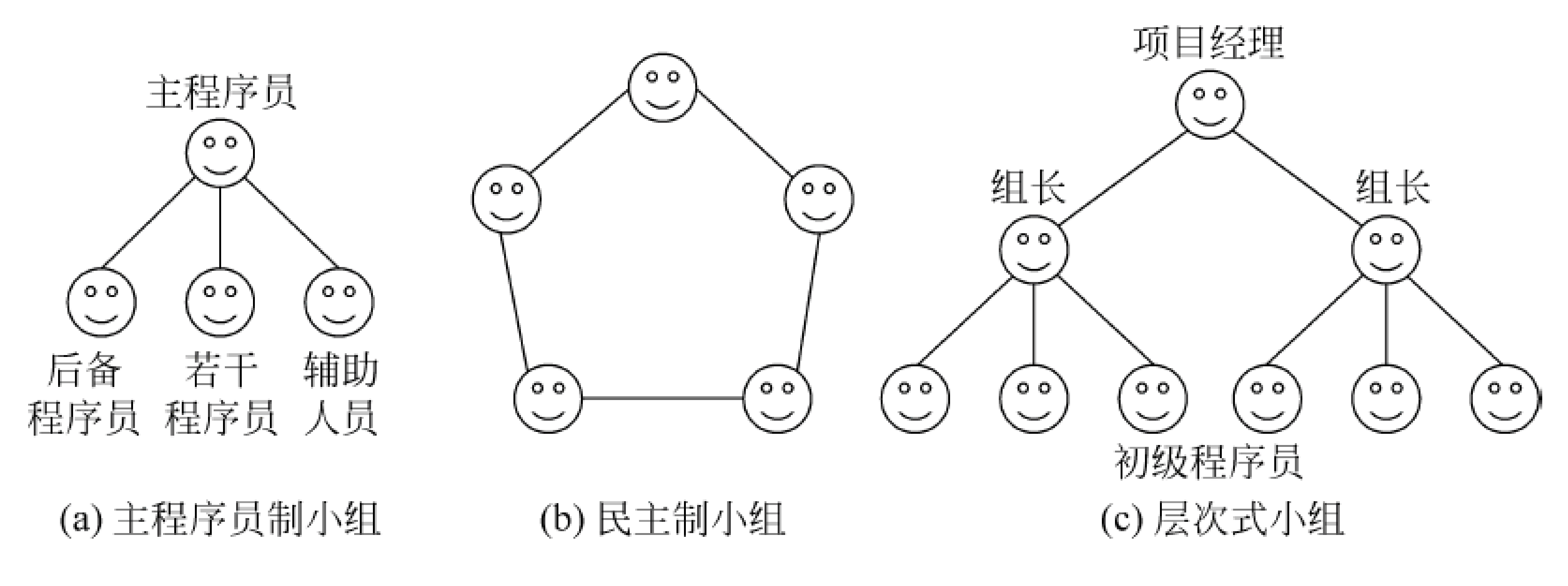


图15.2　项目组的组织结构

1. 主程序员制小组

IBM公司在20世纪70年代初开始采用主程序员制小组，主要是出于下述几点考虑：

(1) 软件开发人员多数缺乏经验。

(2) 程序设计过程中有许多事务性工作，如大量信息的存储和更新。

(3) 多渠道通信很费时间，降低程序员的生产率。

主程序员制小组的组织结构如图15.2(a)所示。小组由一位主程序员、2～5位程序员、一位后备程序员、配置管理及其他辅助人员组成。主程序员制小组的人员组成结构如图15.3所示。

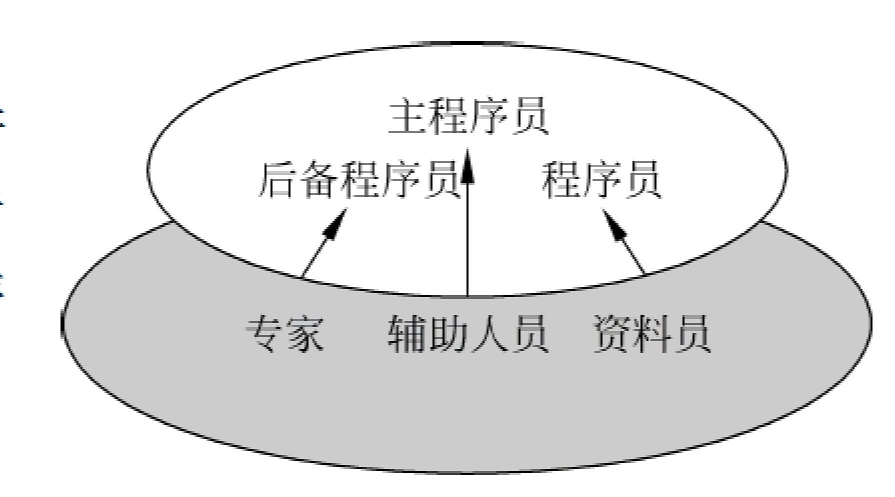


图15.3　主程序员制小组的人员组成结构

人员分工描述如下：

(1) 主程序员。主程序员既是成功的管理人

员,又是经验丰富、技术好、能力强的高级程序员,负责体系结构设计和关键部分的详细设计,负责小组全部技术活动的计划、协调与审查工作,并指导其他程序员完成详细设计和编码工作。

(2) 后备程序员。后备程序员也是技术熟练而且经验丰富的人员,协助主程序员并且在必要时接替主程序员工作。后备程序员也应对项目有深入的了解,在开发过程中的主要工作是设计测试方案、分析测试结果以及独立于设计过程的其他工作。

(3) 程序员。程序员完成详细设计、编码、单元测试等工作。

(4) 其他人员。专家,负责对开发过程中的难点问题给予支持。辅助人员,负责后勤保障工作。资料员,完成与项目相关的事务性工作,如维护项目资料库、项目文档等。

主程序员制小组方式强调主程序员的领导作用,以及与其他技术人员之间的直接联系,简化了人与人之间的沟通。这种组织形式的成功很大程度上取决于主程序员的管理才能和技术水平。

2. 民主制小组

民主制小组如图 15.2(b)所示。民主制小组中也要设置一位组长,但小组成员完全平等,享有充分民主,制定工作目标及做出决策都由全体成员参加。遇到问题时,组内成员之间平等地交换意见,通过协商做出技术决策。虽然也有一位组长,但和组内其他成员完成同样的工作,工作讨论、成果检验都公开进行。

民主制小组的人数不能太多,通常以 2～8 人为宜。由于成员之间的通信是平行的,人数过多时,通信时间过长,而且不同人员设计程序的接口过于复杂。小组规模小,不但可以减少通信问题、接口简单,而且可以共同制定、遵守质量标准,组员之间关系密切,能够互相学习。

这种组织形式强调发挥小组每个成员的积极性,要求每个成员充分发挥主动和协作精神。其优点是小组有高度凝聚力,组内学习氛围浓厚,有利于攻克技术难关。其缺点是没有明确的权威指导开发工作,组员间缺乏必要的协调,最终可能导致项目失败。民主制小组适合于研制时间长、开发难度大的项目。

3. 层次式小组

由于小组成员人数不宜过多,当软件项目规模较大时,应该把程序员分成若干个小组,采用层次结构,如图 15.2(c)所示。在层次式小组中,组内人员分为三级:项目经理(项目负责人)一人负责全面工作,直接领导若干个小组长(高级程序员),每位小组长管理若干个程序员。软件项目开发是作为一个整体在项目经理的指导下进行的,程序员向组长汇报工作,组长向项目经理汇报工作。当产品规模更大时,可以适当增加中间管理层次。

这种组织结构是把主程序员制小组和民主制小组结合起来的一种方法,有利于形成畅通的沟通渠道,充分发挥每个程序员的积极性和主动性,集思广益攻克技术难关。

这种组织结构的特点比较适合层次结构状的课题,可以按组织形式划分课题,然后把子项目分配给基层小组,由基层小组完成。对于大型项目,可以通过层次式将项目划分成若干层。因此,大型软件项目开发比较适合这种组织方式。

15.3.2 各阶段人员需求

大型软件项目的工作量分布情况如图15.4所示。开发阶段(系统定义、功能设计与规格说明、系统开发)工作量占总工作量的40%左右,维护与提高性能阶段(系统测试、安装、试运行、维护)工作量占总工作量的60%左右。图中未画出维护阶段的工作量曲线。

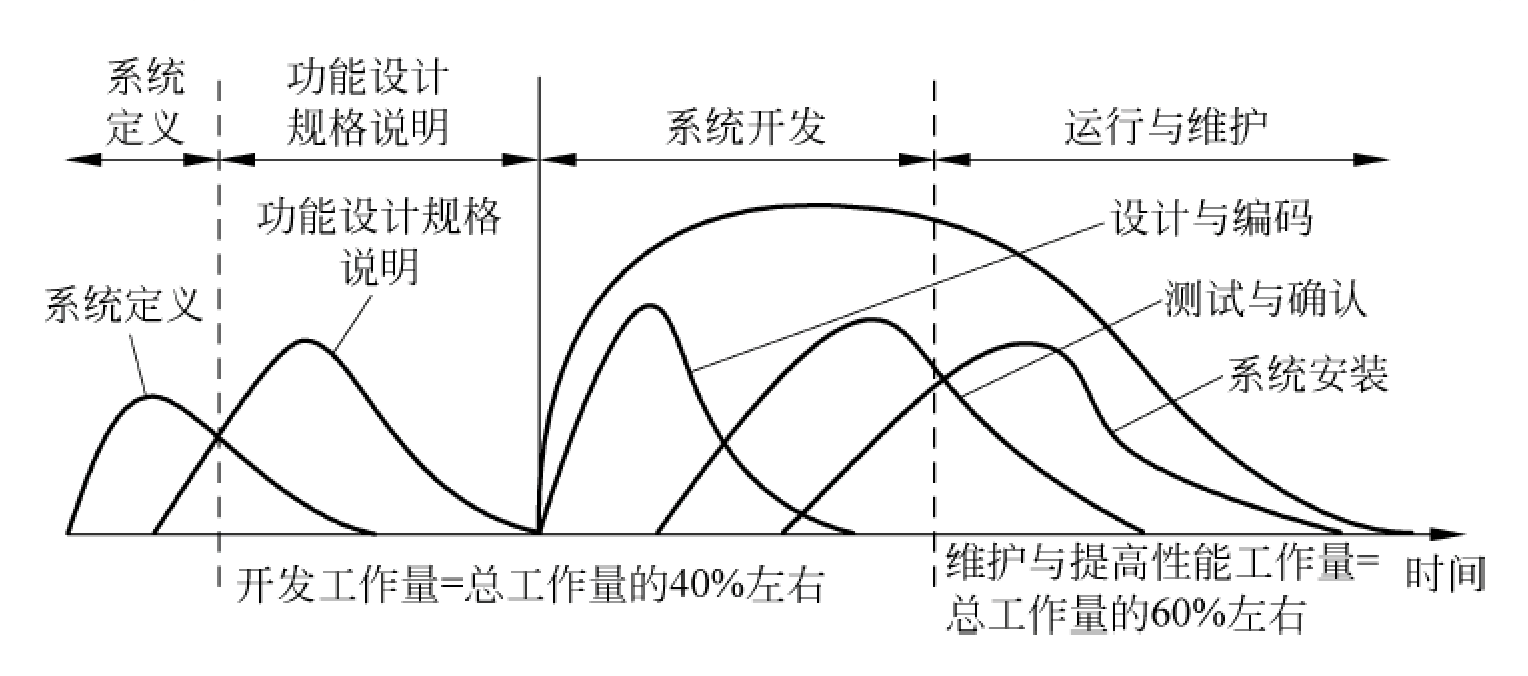


图15.4 大型软件项目的工作量分布情况

根据软件项目的工作量分布图,得到管理人员与技术人员的参与情况,如图15.5所示。在项目开始阶段和结束阶段,管理人员需要做大量工作。有关技术性工作,在开始阶段和结束阶段,高级技术人员参与较多,在中间阶段,初级技术人员参与较多。

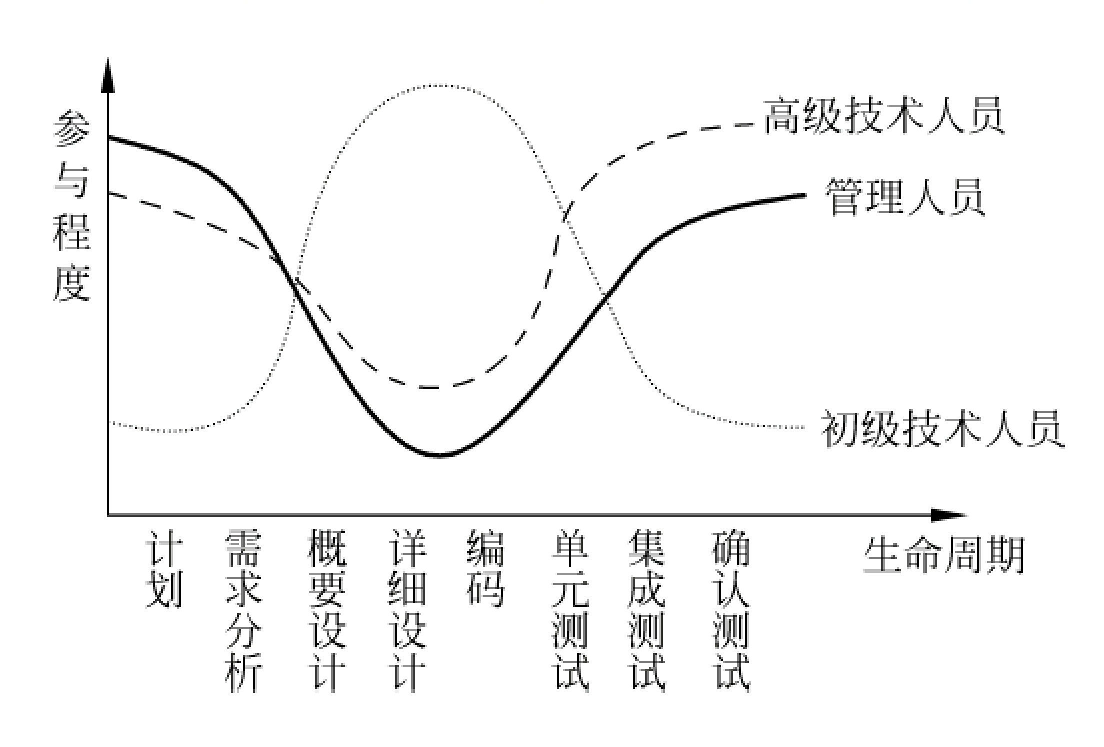


图15.5 管理人员与技术人员的参与情况

软件项目开发实践表明,软件开发各个阶段需要的技术人员类型、层次和数量不同。

(1) 计划与分析阶段:只需要少数人,主要是系统分析员、从事软件系统论证和概要设计的高级软件工程师、项目管理人员等。

(2) 概要设计阶段:需要增加一部分高级程序员。

(3) 详细设计阶段:需要增加软件工程师和程序员。

(4) 编码和测试阶段:需要增加程序员、软件测试员。

在上述过程中,软件开发管理人员和各类专门人员逐渐增加,直至测试阶段,软件项目开发人员的数量达到顶峰。

软件运行初期,参加维护的人员比较多,过早解散开发队伍会给维护带来意想不到的困难。软件运行一段时间后,由于开发人员参与纠错性维护,软件出错率会很快衰竭,这时开

发人员可以逐步撤出。如果系统不作适应性和完善性维护，需要的维护人员就会更少。

在软件开发过程中，人员的选择、分配和组织，是涉及软件开发效率、软件开发进度、软件开发过程管理和软件产品质量的重大问题，必须引起项目经理的高度重视。

15.4 人力资源计划

在项目开发中，经常遇见这样的问题：人越多越好吗？当项目进度延迟时，能否通过增加人力投入来追赶进度？效果如何？会不会越帮越忙？这些是人力资源计划要处理的问题。

在软件项目中，提高人员的素质和效率，科学地组织人员，按照需要来制定人力资源计划，是圆满完成开发工作的重要因素。制定人力资源计划主要基于工作量和进度预估。工作量与项目分时间的比值就是理论上需要的人力数，但选取和分配人力有许多值得研究的问题。本节将在研究理论的基础上给出实例。

15.4.1 人力资源计划理论基础

1. Rayleigh－Norden 曲线

以 Rayleigh 爵士名字命名的曲线原本是用来解释某些科学现象的。1976 年，著名学者 Putnam 把这条曲线与软件开发联系起来，发现在软件生命周期内各个阶段需要的人力资源，具有与 Rayleigh 曲线十分相似的性质。

Rayleigh-Norden 曲线如图 15.6 所示，可用做软件项目不同开发阶段的人力资源分配经验模型。

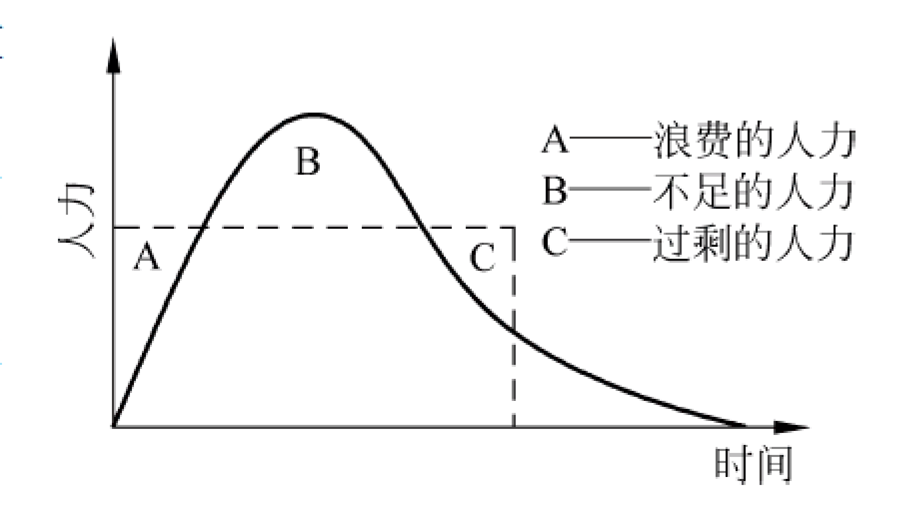


图 15.6 Rayleigh-Norden 曲线

绘制方法为：项目开始为原点，横坐标表示项目进度时间，纵坐标表示某个时间点上需要的人力资源，曲线下方的区域面积就是整个软件项目需要的工作量。

这样绘制的曲线是两头低中间高的。一些大型软件项目数据绘制的曲线显示：最高点左边大致相当于软件的计划与开发时间，右边相当于运行与维护时间，最高点左右两边的工作量之比大致是 4 比 6，也就是说，维护工作量比开发工作量大。

虚线画出的矩形显示了平均使用人力所形成的问题：开始阶段人力过剩，造成浪费(A)；到开发后期需要人力时，又显得人力不足(B)；以后再来弥补，为时已晚(C)，以至可能如 Brooks 定律所说，导致越帮越忙的结果。

2. Putnam 模型

Putnam 在研究 Rayleigh 曲线的基础上，提出 Putnam 模型，用公式表示为：

$$E = L^3/(C_k^3 \times t_d^4) \tag{15.1}$$

式中，E 表示工作量，L 表示源代码行数，C_k 表示技术状态常数，t_d 表示开发时间。工作量的单位是人年，进度的单位是年。从公式中可知，软件开发项目的工作量(E)与开发时

间(t_d)的 4 次方成反比。这说明,通过增加人员来缩短开发时间会带来很多额外的工作量,人员与进度之间是非线性的替代关系,开发过程中人员与时间的折中是十分重要的问题。Putnam 将这一结论称为“软件开发的权衡定律”。

3. Brooks 定律

曾担任 IBM 公司操作系统项目经理的 F. Brooks,从大量的软件开发实践中得出了另一条理论:“向一个已经拖延的项目追加开发人员,可能使项目完成得更晚。”鉴于这一发现的重要性,许多文献称之为 Brooks 定律。Brooks 从另一个角度说明了“时间与人员不能线性互换”这一原则,也就是说,增加人员不一定会缩短时间。

对 Putnam 模型和 Brooks 定律的合理解释是:当开发人员以算术级数增长时,人员之间的通信将以几何级数增长,从而可能导致“得不偿失”的结果。一般说来,由 N 个开发人员组成的小组,要完成既定的工作,相互之间的通信路径总数为 $N\times(N-1)/2$,而通信是需要时间的。所以,当新的开发人员加入项目组后,原有的开发人员必须向新来的人员详细讲解活动或工作包的来龙去脉。并且由于软件开发具有较强的个人风格,所以交流沟通的时间更容易拉长,而后来者还不一定能达到原来开发人员的工作质量。

15.4.2 人力资源计划实例

经验表明,软件项目人力分配大致符合 Rayleigh-Norden 曲线的分布,呈现出前后用人少、中间用人多的不稳定需求情况。但是需要软件开发技术人员时未必能马上获得,所以制定人力资源计划时,要在基本按照曲线配备人力的同时,尽量使某个阶段保持稳定,并且确保整个项目工期人员的波动不要太大。这也就是通常所说的“人力资源计划平衡”。

人力资源平衡法是制定进度计划中使人力资源需求波动最小化的一种方法,尽可能均衡地利用人力资源并满足项目要求完成的进度。人力资源平衡是在不延长项目完工时间的情况下,建立人力资源均衡利用的进度计划。

为了说明人力资源计划平衡的方法,下面举例具体说明。现有某项目已经立项,由于系统较小,准备采用原型法开发,并拟订了一个带有活动工期和人力需求的网络图,如图 15.7 所示。假设参加这个项目的所有成员都是多面手,也就是说,项目成员之间可以相互替代。

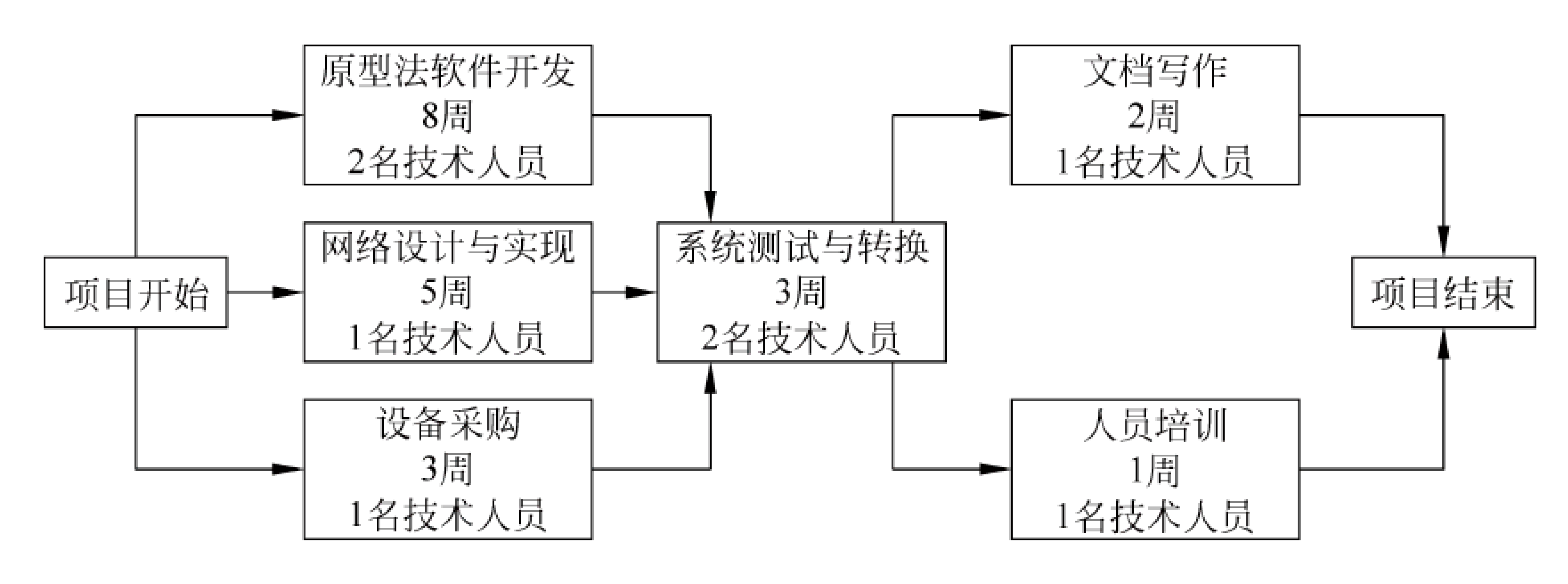
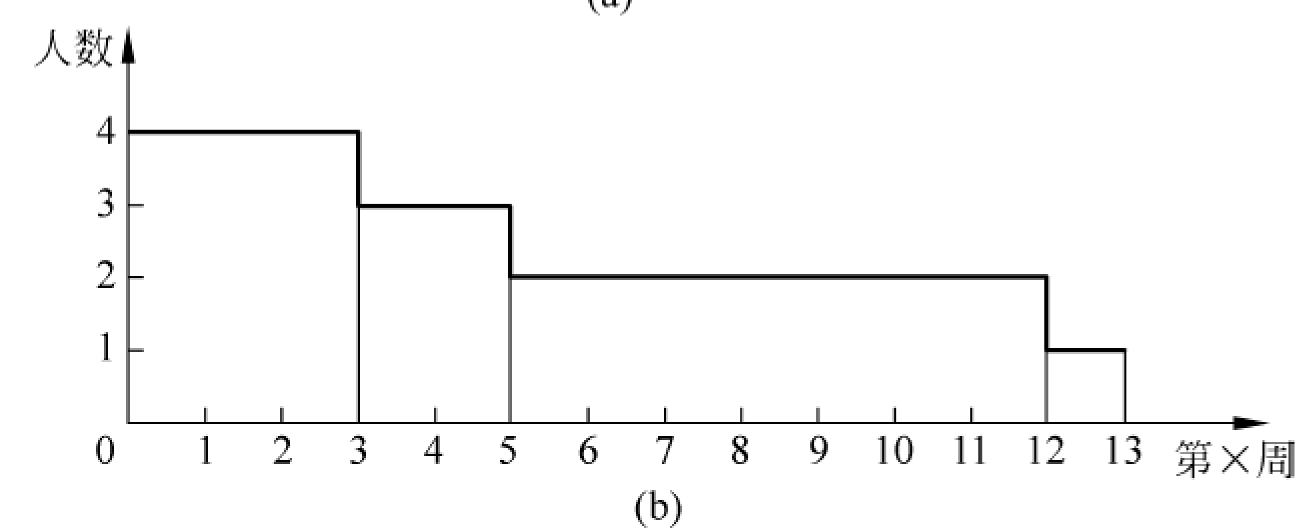


图 15.7 某软件项目人力资源需求网络图

如果不采用项目管理方法,一般人们都希望各项活动尽早开始、尽早结束。现假设网络图中每一活动在最早开始时间执行,绘制相应的人力资源分配图如图 15.8 所示。



(a)



(b)

图 15.8 基于活动最早开始时间的人力资源分配图

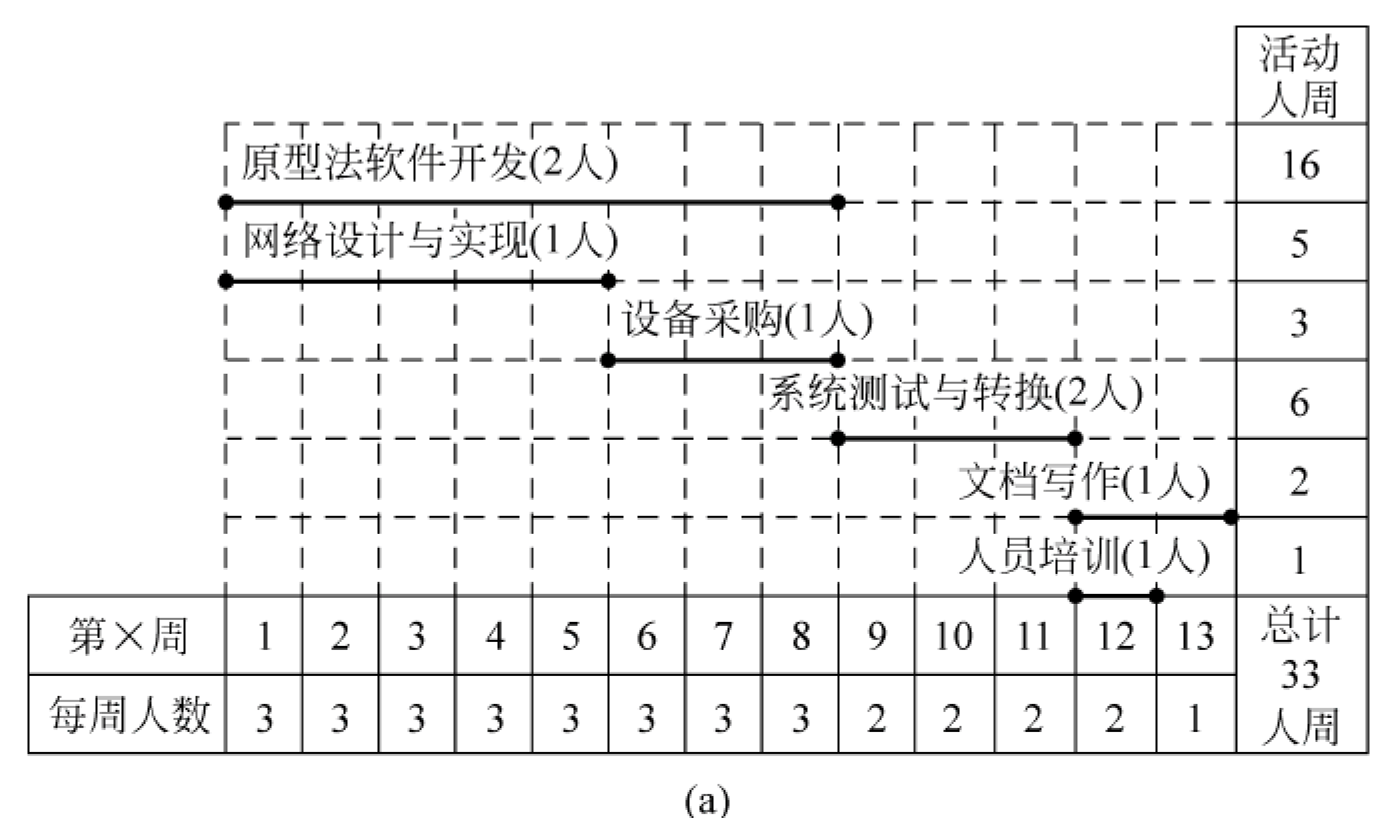
从图 15.8(a)中可以看出,开发该项目共需要 13 周时间,工作量为 33 人周;从图 15.8(b)中可以看出,前三周需要 4 名技术人员,第 4、5 周需要 3 名技术人员,第 6～12 周需要2 名技术人员,第 13 周需要 1 名技术人员,项目人力资源需求波动较大。

为了使人力资源尽可能平衡,研究该项目的人力资源需求网络图,从图中可以看出,该项目的关键路径是"原型法软件开发→系统测试与转换→文档写作"三项活动,其他活动都处于非关键路径上。因而可以将设备采购活动推迟到第 6 周开始,这样就得到调整后的人力资源分配图,如图 15.9 所示。

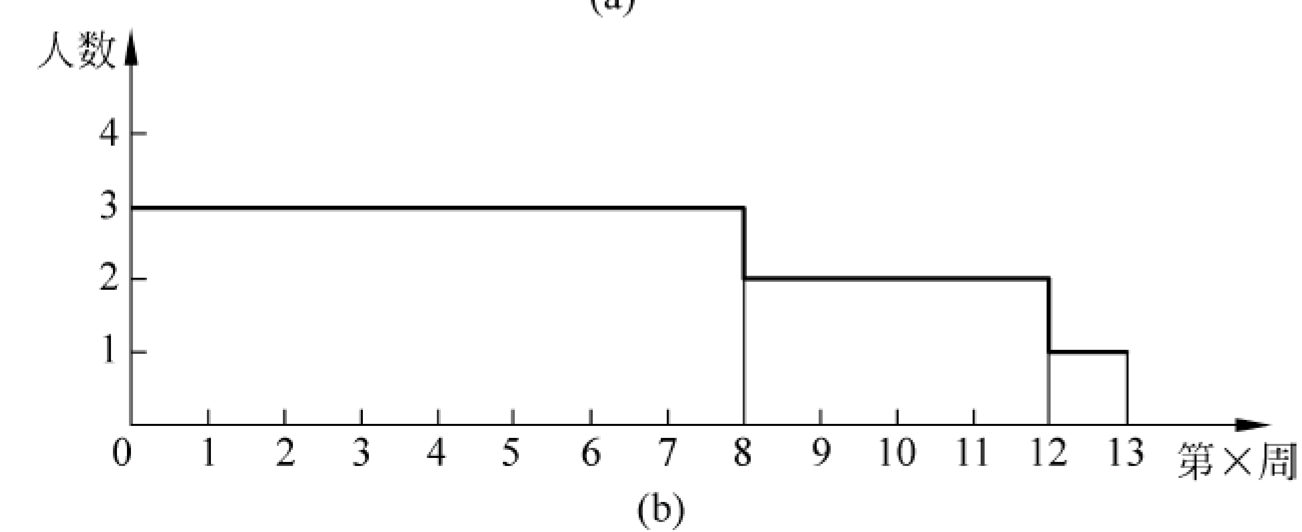
从图 15.9(a)中可以看出,开发该项目还是需要 13 周时间,工作量仍为 33 人周,也就是说,虽然调整了人力资源分配,但并未影响进度;从图 15.9(b)中可以看出,前 8 周需要 3 名技术人员,第 9～12 周需要 2 名技术人员,第 13 周需要 1 名技术人员。相对图 15.8(b)来讲,调整后该项目的人力需求波动较小。

这里需要解释的是,由于采用原型法开发项目,系统调研、原型制作和原型改造都在项目前期进行,需要的人力较多,所以是直接从 Rayleigh-Norden 曲线分布的中部开始,从这个意义上说,本项目的人力使用也基本遵守上述曲线的分布。

上面的例子是在资源没有约束的情况下讨论的,如果资源有约束,比如项目只有两名技术人员,那么在这种情况下进行人力平衡,方法是同样的,也就是通过推迟非关键路径上的活动使资源需求尽可能平衡。不过,进度可能就会有较大的变化,比如上述项目工作量为 33 人周,如果两个人开发,则至少需要 16.5 周才能完成,显然大于 13 周的计划进度。



(a)



(b)

图 15.9　基于人力资源计划平衡的人力资源分配图

15.5　项目经理

要组建优秀的项目团队，必须选择优秀的项目经理。项目经理是项目干系人各方协调配合的桥梁和枢纽，处在项目各方的核心位置。在项目管理过程中，各方面的干系人之间不可避免地产生矛盾和冲突，这些矛盾和冲突需要人来沟通、协商，解决这些问题的关键人物就是项目经理。项目经理是项目的全权负责人，对项目进行全权管理，对项目目标的实现负有主要责任，对项目起到至关重要的作用，是任何人都难以取代的。Boehm 称："一支领导能力出色、管理水平上乘的程序员和分析员队伍的生产效率是一般队伍的四倍。"这个结论证明了领导才能和项目管理技能对取得高效软件开发成就的重要性，因为软件开发团队的工作能力在很大程度上取决于团队领导——项目经理。没有有效的项目管理就不会有项目的成功，而没有合格的项目经理就不会有高效的团队，就不会有软件开发项目的成功。

15.5.1　项目经理的技能要求

软件项目经理不能只具有必要的项目管理技能，而是需要具有过硬的专业知识、丰富的管理经验。

对软件项目经理的技能要求是：在软件行业中某一技术领域具有权威，技术过硬，任务分解能力强；注重对项目成员的激励和团队建设；能良好地协调项目小组成员的关系；具备较强的客户人际关系能力；具有很强的工作责任心；能够经常接受加班的要求；将自己

定位为既是管理人员又是技术人员。软件项目经理通常的技能要求如图 15.10 所示。

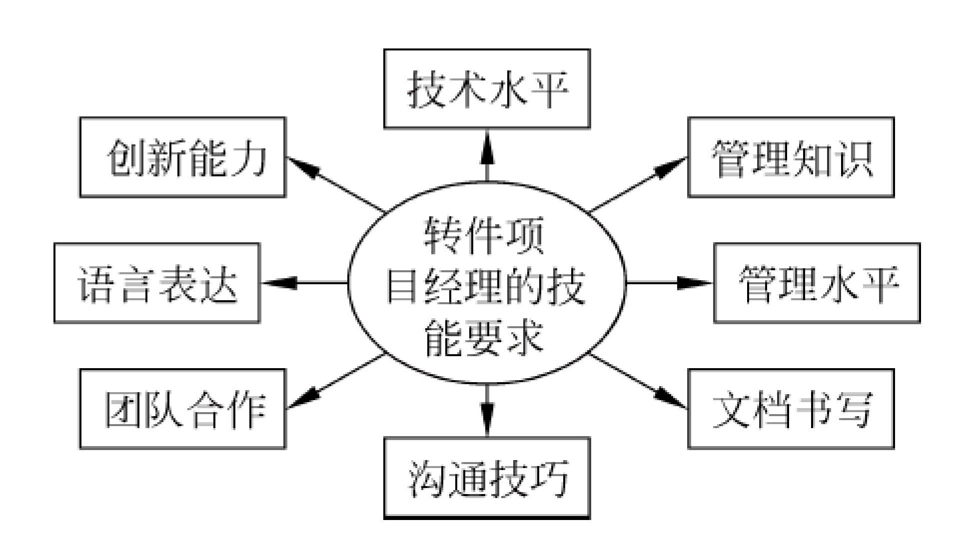


图 15.10　软件项目经理的技能要求

15.5.2　项目经理的素质与职责

1. 应具备的素质

作为一个合格的项目经理，应具备的素质如下：

(1) 高尚的品德。品德高尚、以德服人，以修养和品德感染人，勇于承担责任、乐于助人，使团队具有向心力，从成功走向更大的成功。

(2) 执著力。软件开发技术上可能遇到各种困难，推行各种规范和管理制度可能遇到各种阻力和障碍，要有应付挫折的思想准备，有坚定的信念，有坚韧不拔的精神，才能成功。

(3) 智力素质。软件开发需要不断地解决问题，管理需要思路敏捷、视野开阔，这要求项目经理具有较高的智力素质，并通过经验积累和实践过程不断提高智力素质。

(4) 亲和力。要融于团队之中，被每个成员信任，关心下属的工作和生活，主动与下属沟通，为下属争取合法权益，做下属的知心朋友。

(5) 责任心。项目经理对项目的成败负主要责任。项目经理要有高度的责任心，把项目的各项工作落到实处，认真检查每个成员的工作。

(6) 善于总结。不断地总结成功经验和失败教训，听取别人意见，博采众长，在总结中不断地提高和完善自己。

2. 应履行的职责

项目经理作为软件项目管理的负责人，在开发过程中起着举足轻重的作用。其职责主要有以下几个方面：

(1) 审核软件项目的立项文档。立项文档的撰写工作通常由营销部门负责，主要包括经公司评审并批准的立项建议书、下达指令性的任务书、签订的合同书或委托书(订单)等。项目经理要仔细分析立项文档的内容并进行审核。

(2) 细化软件开发计划，实施任务分配。项目经理根据立项文档制定初步的软件开发计划，需求分析完成后，再修改并细化软件开发计划。软件管理部门对软件开发计划进行评审，评审通过后，由项目经理根据人员计划，对项目组成员进行具体分工。

(3) 结合用户需求报告完善项目计划。系统分析师通过调研，获取用户需求，写出需求分析报告，经用户确认并签字后，作为验收测试的依据。管理部门对需求分析报告进行评审后，项目经理根据需求分析报告修改开发计划，并对修改后的开发计划进行评审与冻结。

(4) 配合系统设计师完成系统设计。系统设计分为概要设计和详细设计。系统设计完成后,管理部门对概要设计和详细设计文档进行评审,评审通过后,作为项目的编程基线。项目经理在系统设计过程中,应当辅助系统设计师完成相关工作。

(5) 组织程序员编写代码与调试。根据系统设计文档,项目经理组织编程人员进行代码实现、单元测试和集成测试等相关工作,并将此过程中出现的问题及时与分析师、用户进行沟通。

(6) 组织项目组成员书写相关手册。调试工作完成后,项目经理组织人员编写用户指南(使用手册、安装手册)。根据需要,还可能编写系统维护手册和其他有关培训手册,并对相关人员进行培训。对产品进行包装,形成公司对外发布和保存管理的版本。

(7) 完成项目报告和总结工作。软件项目内部验收或用户验收完毕后,项目经理应召开项目工作总结会,书写项目总结报告。应从企业文化、积累经验、技术创新等方面进行全面总结,向软件管理部门提供详细资料,由管理部门将此资料追加到软件过程数据库中。

15.6 团队建设

在软件项目开发过程中,拥有一支有能力、有经验的开发队伍是项目成功的关键。如果人员组织形式和配置方法不当,会影响到软件开发乃至后期维护。如果项目开发时间长,开发人员工作量不饱满,会使开发成本增高、效益降低;如果项目开发时间短,项目组人员配置不足,会造成开发人员长期加班加点,开发工作质量不高。因此,确立一套完整、科学、可操作的工作方法,对项目团队的人员构成、组织结构、人数安排等进行合理配置,是公司管理层必须考虑的问题。

软件项目开发团队是开发过程中紧密协作、并肩作战、相互负责的一个群体。团队成员拥有共同的工作目标、高度的凝聚力、有效的沟通、合理的分工与协作。项目经理带领团队成员在规定的时间、费用范围内,完成软件项目开发工作。

15.6.1 团队建设的重要性

团队建设的重要性体现在两个方面:

(1) 和谐的工作环境是企业发展的关键。人是企业发展的首要因素,进行团队建设就是要为团队成员提供和谐的工作环境。如果工作和生活在风清气正、团结和谐、相互信任的环境里,就会心情舒畅、精神愉悦,浑身有使不完的劲儿,就可以集中精力做事情、一心一意干事业。反之,如果工作和生活在一个矛盾重重、关系紧张、彼此猜忌的环境里,就会心情压抑、精神沮丧,难以凝神聚力地投入工作。软件工程是针对人的工程,而不是针对技术的工程,管理者要深刻理解马斯洛的需要层次理论,看到在软件研发过程中人员多方面的需求,加强员工凝聚力,给员工发挥智慧和才能的空间。

(2) 团队建设是软件项目成功的重要因素。团队工作是整体配合的过程,而不是单打独斗的过程,是充分发挥个人能力并融合到整体的过程。据 Standish Group 公司调查统计(见表 15.1),在软件项目开发不成功的原因中,团队组织问题、缺乏协调、沟通不良、信息不畅通等涉及软件项目研发的软性问题占到了 60%以上。软件项目不成功的因素之一也是

由于团队组织建设出现问题造成的。因此，在软件项目开发过程中，在关注项目进度、费用、质量等硬指标的同时，更应重视团队建设。

表 15.1 软件项目研发不成功的原因

需求定义	问题比率(%)	团队组织	问题比率(%)	项目运行	问题比率(%)
定性不明确	15.1			缺乏协调	8.8
进度安排不当	9.8	人员配置不当	12.2	控制不良	7.1
决策不力	8.6	职责不明	6.2	沟通不良	6.5
信息不良	3.9	负责人不得力	4.6	领导不力	5.8
变化因素	5.6			敬业精神差	5.8
合计	43.0	合计	23.0	合计	34.0

15.6.2 团队建设过程

团队建设过程如图 15.11 所示，包括组建、磨合、正规、成熟和结束五个阶段。

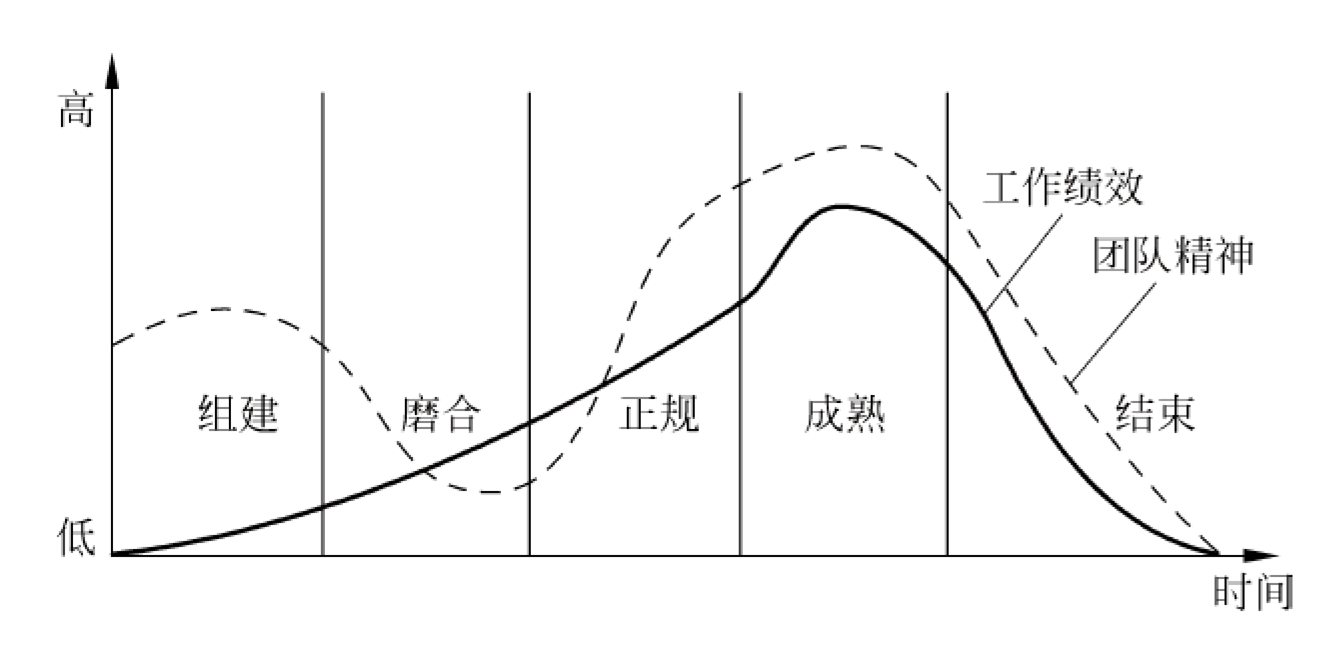


图 15.11 团队建设过程

第一阶段：组建团队。接到项目开发任务后，需要确认项目经理，然后根据项目开发内容、工作量估计等确认项目团队成员的组成及数量，组建项目团队。项目团队成立后，项目经理召开团队会议向团队成员说明项目任务、目标、规模、人员组成、规章制度和行为准则，明确每个人的岗位和责任，建立团队与外界的初步联系及相互关系，确定团队的权限，建立团队的绩效机制。同时，积极争取公司各方面的支持，收集有关项目信息。

第二阶段：团队磨合期。刚刚组建的项目团队，由于成员来自不同的部门、不同的工作岗位，相互之间不熟悉，不了解各自特长，工作刚开始会出现成员之间相互配合少的现象，工作之间会出现互相推诿、扯皮等现象，成员之间容易出现冲突，这就需要项目经理来协调，成员之间要取长补短、互相学习。项目经理的主要任务是树立威信，建立切实可行的工作规范标准，多与成员沟通交流，以支持成员工作为主，做研发工作问题的发现者、解决者。

第三阶段：团队走向正规。经过一段时间的磨合，成员之间相互熟悉、信任程度逐步增强，内部的沟通交流与合作逐渐增多，项目团队整体水平增强，工作效率明显提高。项目团队经过磨合期后，团队成员之间的协调关系已经确立，项目有关规程得以改进和规范，项目经理应逐渐下放权力，对研发过程中出现的问题与公司领导、涉及的部门及成员进行交流，提出解决问题的方法，及时解决研发过程中存在的问题，保证各项工作按计划进行。

第四阶段：团队进入成熟。随着工作进展，团队成员之间越来越熟悉，交流沟通及时、信息传递流畅，表现为项目成员之间的相互信任和默契配合，当某一成员有困难时，大家会伸出援助之手帮助解决，当遇到难题时大家一起开动脑筋想办法解决，项目实施进入高效阶段。项目组成员拥有共同的目标，能更快速、更容易地到达目的地，因为彼此之间能相互推动，以合作取代竞争，一起创造项目整体的工作价值。在这一阶段，项目经理要授予团队成员充分的权力，允许个人或小组进行工作流程、工作方法等方面的创造性工作。

第五阶段：项目结束。软件项目在规定的时间、费用、质量范围内完成了，团队的目标基本实现，项目组成员要在时间、成本、质量控制等方面总结经验教训。软件项目工作结束，团队成员要回到原来各自的工作岗位。在实施工作过程中，成员之间增进了友谊、加深了感情、积累了经验，为做好下一个软件项目打下了基础。

15.6.3　打造高效团队的策略

要打造高效团队，需要做到以下几个方面：

(1) 称职的项目经理。项目经理是团队的领导者，直接对项目开发过程中的各项工作负责。要创建一个互相信任、具有高度凝聚力、民主气氛浓厚的团队，需要项目经理具有良好的领导组织和协调能力、较高的专业技术水平和良好的职业素养，善于交流沟通，及时、妥善地处理项目开发中出现的各种问题。

(2) 目标明确、分工合理。一是要明确全部项目工作，把复杂的工作逐步分解成要素工作，使得成员明确具体工作，容易操作和控制；二是使用工作责任分配矩阵，对团队成员进行分工，直观地反映出每个成员的职责。可以详细列出每项工作的具体负责人及完成时间、工作内容，使每项具体任务都落实到团队成员个人，确保项目开发过程中事事有人做，人人有事干。

(3) 良好的交流、合作和创新精神。加强项目团队的内部交流、合作和创新，提高项目质量，保证按期完成。一是要信息交流畅通，通过交流拓宽思路、统一思想、达成共识；二是要增强相互之间的协调合作，每个成员按照工作标准在做好自己工作的同时，善于和同事配合，提高整个项目的质量和效率；三是要敢于创新，没有完全相同的软件项目工作，要善于学习、善于借鉴其他项目开发工作中好的做法，通过创新提高项目研发的成功率。

(4) 打造学习型团队。面对快速发展的 IT 技术、不断变化的市场需求，需要成员树立三种学习理念：一是树立学习是生存和发展需要的理念，学习是为未来投资，是为了自己的生存和发展；二是树立终生学习的理念；三是树立“在工作中学习，在学习中创新，在创新中发展”的理念，把学习引入到工作中，使学习与工作有机结合。同时做好内部培训工作，与公司其他部门、外界软件公司进行技术合作交流，举办专题讲座、学术研讨会等。

(5) 合理的团队绩效考核。对项目团队中每个成员考核是必要的，考核的目的就是总结分析项目研发过程中存在的优缺点，促进工作，而不是处罚和批评。因此，合理的团队绩效考核非常重要。技术人员长期在有压力的环境下工作，合理是指不能超出成员承受的极限，否则就会身心疲惫不堪，导致工作效率和质量下降。

思考题

1. 软件项目人力资源有怎样的行业特征?
2. 软件项目人力资源管理的主要内容是什么?
3. 项目组内部人员的组织形式有哪几种?主程序员制小组人员如何分工?
4. 理解大型软件项目各阶段人员工作量分布以及人员需求情况。
5. 了解人力资源计划的理论基础。
6. Brooks 定律说明了什么问题?
7. 掌握运用人力资源需求网络图制定人力资源计划的方法。
8. 对软件项目经理的技能要求有哪些?
9. 项目经理应具备怎样的素质?应履行哪些职责?
10. 简述项目团队建设的重要性。
11. 团队建设通常经过哪些阶段?每个阶段有何特征?
12. 打造高效团队的策略有哪些?

实验篇

软件工程实验课的目标，是通过实际操作使学生掌握较强的软件开发的组织、管理、实施的方法和技术，促进学生掌握软件工程的理论知识、标准和规范。通过实践，培养学生工程化管理软件开发的能力，包括可行性研究、需求分析、系统设计、系统实现和维护、软件项目管理等。在实际的软件开发中，这些能力主要通过文档书写来体现。实验篇安排软件开发过程中最重要的10个文档，来培养学生的综合能力。

由于每个完整的文档涉及的内容都很多，很难在有限的实验课时内完成，学生可根据实际情况，对每个文档所列标题有选择地完成。



可行性分析报告

可行性分析报告(Feasibility Analysis Report,FAR)又称可行性研究报告,是项目初期策划的结果。可行性分析报告分析项目的要求、目标和环境,提出几种可供选择的方案,并从技术、经济和法律等各方面进行可行性分析。可行性分析报告可作为项目决策的依据,也是项目建议书、投标书等文件的基础。

1. 项目概述

简述文档适用的项目和软件的用途,描述项目和软件的一般特性;概述项目开发、运行和维护的历史;标识项目的投资方、需方、用户、开发方和支持机构;标识当前和计划的运行现场。

2. 可选择的方案

1) 原有方案的优缺点、局限性及存在的问题

说明原有方案的优点、缺点、局限性和存在的问题。

2) 可选择的系统方案 1

说明可选择的系统方案 1,其中包括方案概述、处理流程和数据流程、对现有系统的改进、预期影响、局限性等。

3) 可选择的系统方案 2

说明可选择的系统方案 2,其中包括方案概述、处理流程和数据流程、对现有系统的改进、预期影响、局限性等。

……

n) 选择最终方案的准则

说明选择最终方案的评价准则与评价标准。

3. 所建议的系统

1) 对所建议系统的说明

概要说明所建议系统,并说明使用的基本方法、理论依据以及如何满足系统要求。

2) 数据流程和处理流程

给出所建议系统的数据流程和处理流程。

3) 与原系统的比较

若有原系统,则逐项说明所建议系统相对于原系统的改进。

4）影响（或要求）

影响（或要求）包括设备、软件、运行、开发、环境、经费等方面的影响（或要求）。

4. 经济可行性（成本—效益分析）

1）投资

投资包括基本建设投资（如开发环境、设备、软件和资料等）、其他一次性和非一次性投资（如技术管理费、培训费、管理费、人员工资、奖金和差旅费等）。

2）预期的经济效益

预期的经济效益包括一次性收益、非一次性收益、不可定量的收益、收益/投资比、投资回收期等。

3）市场预测

对未来的市场情况进行预测，如潜在的客户数、潜在的市场份额等。

5. 技术可行性

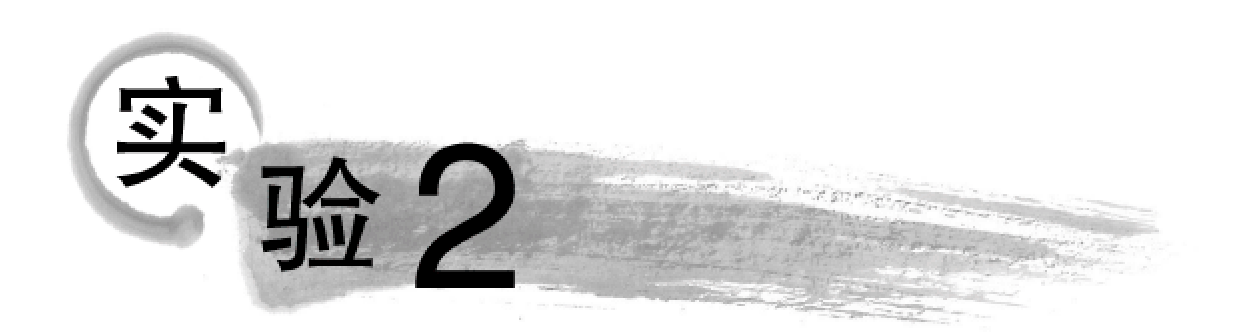
说明现有资源（如人员、环境、设备、技术条件等）能否满足此工程和项目的实施要求。若不满足，则应考虑补救措施（如分承包方参与，增加人员、投资、设备等）。涉及经济的问题应进行投资、成本、效益可行性分析，最后再确定此项目是否具备技术可行性。

6. 法律可行性

说明系统开发可能导致的侵权、违法和责任。

7. 用户使用可行性

说明用户使用方面的可行性。例如，从用户单位的行政管理和工作制度等方面来看，能否使用该系统；从用户单位使用人员的素质和培训来看，能否满足使用该系统的要求等。



软件需求规格说明

软件需求规格说明(Software Requirement Specification,SRS)是描述需求的文档,是软件生命周期最重要的文档。已经确定的需求应当得到清晰准确的描述。同时,为了确切表达用户对软件的输入输出要求,还需要制定数据要求说明书及编写初步的用户手册,着重反映用户界面和用户使用的具体要求。

1. 需求概述

1) 目标

开发本系统的目标。

2) 运行环境

简要说明本系统的运行环境(包括硬件环境和支持环境)的规定。

3) 用户特点

描述本系统用户的类型和特点。

4) 关键点

说明本软件需求规格说明中的关键点(如关键功能、关键算法和关键技术等)。

5) 约束条件

列出进行本系统开发工作的约束条件。如经费限制、开发期限、采用的方法和技术等。

2. 系统数据流图

比较细化的数据流图(用一张图将所有数据流、文件及处理表示出来)。

3. 需求描述

1) 功能需求

软件功能是软件应具有的效能和作用,软件目标要通过软件功能来表达和实现,软件功能也是软件呈现给用户的直接效果。依据软件目标,形成用软件功能模型描述的结果,并定量或定性地叙述对软件提出的功能要求。

2) 性能需求

所开发软件的技术性能指标,主要是精度、时间特性和灵活性等方面的要求。

3) 输入输出要求

描述各输入输出数据类型,并逐项说明其媒体、格式、数值范围、精度等。

4）数据管理能力要求

说明需要管理的文卷和记录的个数以及表和文卷的大小规模。

5）环境要求

软件系统运行时所处环境的要求，包括硬件环境、软件环境、网络环境等。

6）安全保密要求

应当在这方面恰当地做出规定，对开发的软件给予特殊设计，保证在运行过程中的安全保密性能。

7）用户界面要求

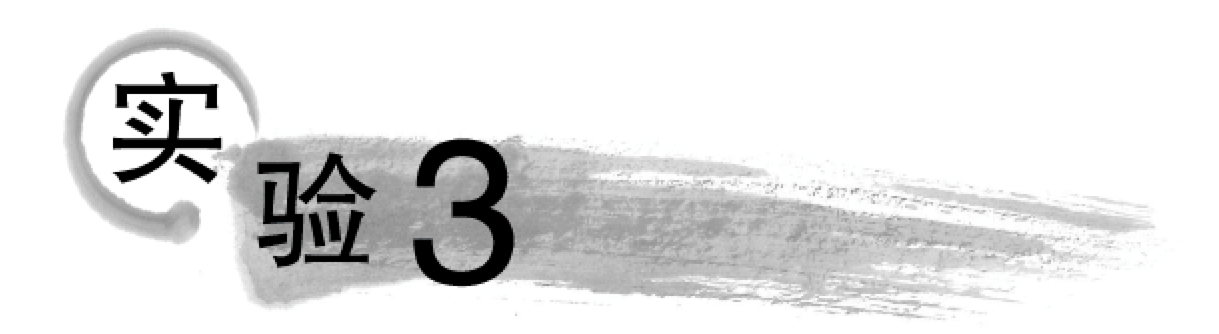
规定用户界面应达到的要求。

8）软件成本消耗与开发进度要求

软件项目立项后，根据合同规定，对软件开发的进度和各步骤的费用提出要求，作为开发管理的依据。

4．尚未解决的问题

说明软件需求中尚未解决的遗留问题。



实验3 软件结构设计说明

软件(结构)设计说明(Software Design Description,SDD)是概要设计阶段的文档,描述计算机软件配置项(CSCI)的设计,包括CSCI级设计决策、CSCI体系结构设计(概要设计)和实现该软件所需的接口设计。SDD向需方提供了设计的可视性,为软件支持提供所需要的信息。

1. 结构设计概述

1) 设计任务

简要描述概要设计的任务。

2) 设计原则

从抽象、逐步求精、模块性和信息隐蔽等方面简要描述概要设计的原则。

3) 设计策略

从以下一些方面描述本系统的设计策略:

(1) 改造软件结构,降低耦合度,提高内聚度。

(2) 减少扇出,追求高扇入。

(3) 使任意模块的作用域在其控制域内。

(4) 降低模块的接口复杂度和冗余度,提高协调性。

(5) 模块功能可预测,避免对模块施加过多限制。

(6) 追求单入口、单出口的模块。

(7) 为满足设计和可移植性要求,把某些软件用包封装起来。

2. 体系结构设计

1) 程序(模块)划分

用一系列图表列出本软件配置项内每个程序(包括每个模块和子程序)的名称、标识符和功能等。

2) 程序(模块)层次结构关系

用一系列图表列出本软件配置项内每个程序(包括每个模块和子程序)之间的层次结构与调用关系。可使用层次图、HIPO图、结构图等工具。

3) 主要模块功能说明

对比较重要的核心模块,描述其功能。

4）全局数据结构说明

说明本程序系统中使用的全局数据常量、变量和数据结构。

常量：数据文件名称、所在目录、功能和具体常量说明等。

变量：数据文件名称、所在目录、功能和具体变量说明等。

数据结构：数据结构名称、功能和具体数据结构说明（定义、注释和取值等）等。

3. 主要接口设计

分条描述软件配置项的接口特性，既包括软件配置项之间的接口，也包括与外部实体（如环境、配置项和用户）之间的接口。如果这些信息的部分或全部已在接口设计说明，或在其他地方说明，可在此处引用之。

1）XX 与 YY 接口

XX 模块与 YY 模块的接口设计。

2）AA 与 BB 接口

AA 模块与 BB 模块的接口设计。

……

n）RR 与 JJ 接口

RR 模块与 JJ 模块的接口设计。

4. 需求的可追踪性

（1）从本文档中标识的每个软件配置项到分配给它的软件配置项需求的可追踪性。

（2）从每个软件配置项需求到它被分配给的软件配置项的可追踪性。

实验4 软件详细设计说明

软件详细设计说明(Software Detailed Design Description,SDDD)是详细设计阶段的文档。详细设计是对概要设计的进一步细化,确定模块的两个内部特性,即描述每个模块的执行过程(怎么做)和定义模块的局部数据结构。

应对软件项的每一软件部件进行详细设计。软件部件应细化到更低级别,这些级别包含能被编码、编译、测试的软件单元。应确保来自这些软件部件的所有软件需求都被分配到软件单元。

1. 详细设计概述

1) 设计任务

简述详细设计的任务。

2) 结构程序设计方法

从结构程序设计的产生与发展、基本控制结构、优越性等方面描述结构程序设计方法。

2. 程序描述

由于程序描述需要设计的内容较多,建议通过对其中的一个模块进行描述,掌握程序描述的方法。

(1) 功能。

(2) 性能。

(3) 输入输出。

(4) 算法及程序逻辑。模块所选用的算法。运用详细设计表示工具,详细描述模块算法的实现。

(5) 接口。

(6) 存储分配。

(7) 测试要点。

给出测试模块的主要测试要求。

3. 主要模块的详细设计

运用详细设计表示工具(流程图、盒图、问题分析图、IPO 图),分别进行不同模块的设计,每种工具设计一个模块,着重理解各种工具的使用方法。

1）A 模块——流程图

（1）模块功能简介及主要设计思路。

（2）画流程图。

2）B 模块——盒图

（1）模块功能简介及主要设计思路。

（2）画盒图。

3）C 模块——问题分析图

（1）模块功能简介及主要设计思路。

（2）画问题分析图。

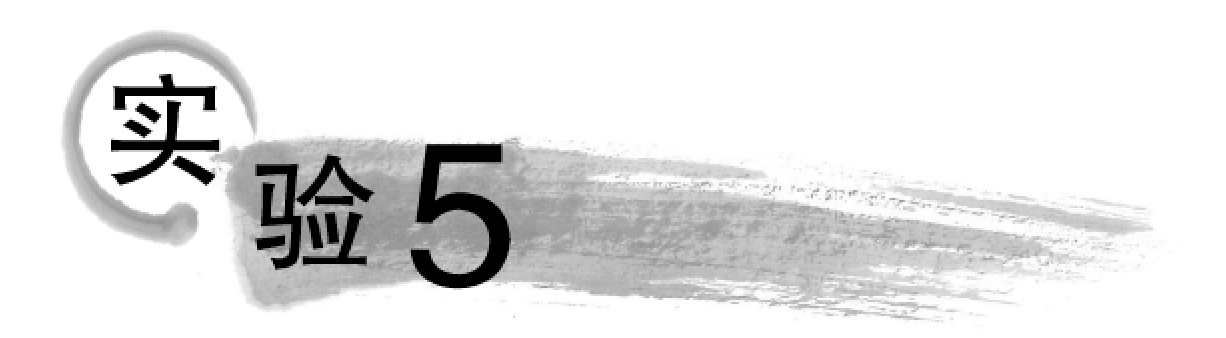
4）D 模块——IPO 图

（1）模块功能简介及主要设计思路。

（2）画 IPO 图。

4. 对详细设计表示工具的总结

结合设计过程，对详细设计表示工具的四种图形（流程图、盒图、问题分析图、IPO 图）进行比较，总结出优点、缺点及适用情况等。



实验5 软件测试报告

软件测试报告(Software Testing Report,STR)是对计算机软件配置项(CSCI)、软件系统或子系统,或与软件相关项目执行合格性测试的记录。通过 STR,需方能够评估所执行的合格性测试及其测试结果。

1. 软件测试方法综述

对软件测试方法的总结性叙述。

2. 详细的测试结果

主要掌握测试方法,要求至少书写两个模块的详细测试结果。

1)(模块 1 的名称)

(1) 测试用例。

设计测试用例。

(2) 测试结果。

综述该项测试的结果。尽可能以表格的形式给出与该测试相关联的每个测试用例的完成状态(例如,“所有结果都如预期的那样”、“遇到了问题”、“与要求有偏差”等)。

(3) 测试分析。

测试过程遇到的问题,分析测试用例/过程的偏差和测试用例存在的问题。

2)(模块 2 的名称)

(1) 测试用例。

(2) 测试结果。

(3) 测试分析。

3. 测试结果概述

1) 对被测试软件的总体评估

应书写以下内容:

(1) 根据本报告中展示的测试结果,提供对该软件的总体评估。

(2) 标识在测试中检测到的任何遗留的缺陷、限制或约束。可用问题/变更报告提供缺陷信息。

(3) 对每一遗留缺陷、限制或约束,应描述:

① 对软件和系统性能的影响,包括未得到满足需求的标识。

② 为了更正它，将对软件和系统设计产生的影响。

③ 推荐的更正方案/方法。

2）测试环境的影响

对测试环境与操作环境的差异进行评估，并分析这种差异对测试结果的影响。

3）改进建议

对被测试软件的设计、操作或测试提供改进建议。应讨论每个建议及其对软件的影响。

4. 测试记录

尽可能以图表或附录形式给出一个本报告覆盖的测试事件的按年月顺序的记录。测试记录应包括：

(1) 执行测试的日期、时间和地点。

(2) 用于每个测试的软硬件配置，(若适用)包括所有硬件的部件号/型号/系列号、制造商、修订和校准日期；所使用的软件部件的版本号和名称。

(3) (若适用)与测试有关的每一活动的日期和时间，执行该项活动的人和见证者的身份。

5. 测试总结

总结主要的测试活动和事件；总结资源消耗，包括人力消耗、物质资源消耗等。



实验6 软件产品规格说明

软件产品规格说明(Software Product Specification,SPS)包含或引用可执行软件、源文件以及软件支持的信息,包括一个计算机软件配置项(CSCI)“已建成”的设计信息和编辑、构造及修改的过程等。

SPS可用于订购可执行软件和对应于该CSCI的源文件,针对该CSCI的基本软件支持文档。注意,不同的组织对软件的订购和移交有着不同的策略,这种策略应在使用这个文档之前决定。

1. 系统概述

概述本文档适用的系统和软件的用途。描述系统和软件的一般特性;概述系统的开发、运行与维护历史;标识项目的投资方、需方、用户、开发方和支持机构;标识当前和计划的运行现场。

2. 文档概述

概述本文档的用途和内容,并描述与其使用有关的保密性和私密性要求。

3. 版本说明

1) 发行材料清单

通过标识号、标题、缩略语、日期、版本号等列出构成发行软件的所有物理媒体(例如列表、磁带、磁盘)和有关的文档。应包括适用于这些项的保密性和私密性要求、处理它们的安全措施(例如对静电和磁场的关注)、关于复制和许可证条款的说明和制约。

2) 软件内容清单

通过标识号、标题、缩略语、日期、版本号等列出构成发行软件的所有计算机文件。应包含适用的保密性和私密性要求。

3) 已安装的变更

应包含一张列表,记录当前的软件版本自上一个版本后引入的所有变更。如果使用了变更类别,则变更应按这些类别进行划分。应标识与每一变更和该变更对系统运行和其他软硬件接口产生的影响的相关问题报告、变更建议和变更通告。本条不适用于最初的软件版本。

4) 适应性资料

应标识或引用包含在软件版本中所有场地专用的资料。对于第一版之后的软件版本,

本条应描述对适应性资料所做的变更。

5）相关文档

应按标识号、标题、缩略语、日期、版本号和发行号，列出与发行软件有关但未包含在其中的所有文档。

6）安装指令

应提供或引用以下信息：

（1）安装该软件版本的指令。

（2）为使该版本可用而必须安装的其他变更标识，包括未包含在软件版本中的场地专用的适应性资料。

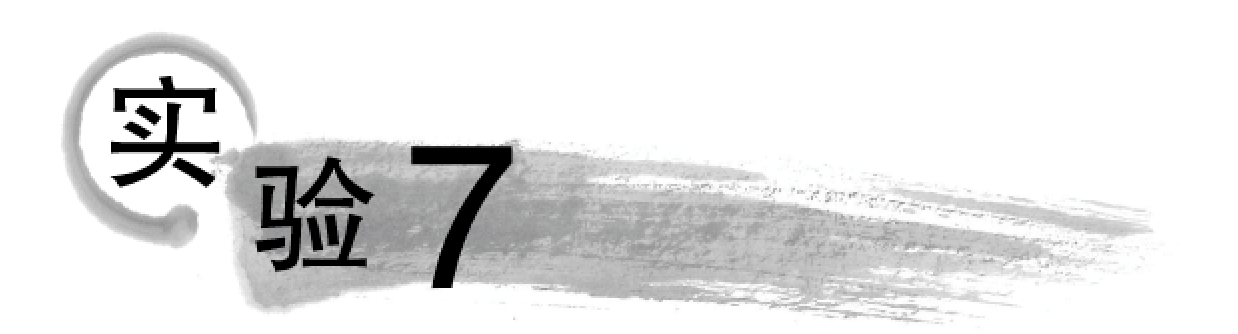
（3）与安装有关的保密性、私密性和安全提示。

（4）判定版本是否被正确安装。

（5）安装中遇到问题后的求助联系方式。

7）可能的问题和已知的错误

应标识软件版本在发行时可能存在的问题或已知的错误、解决问题和错误应采取的步骤，以及给出识别、避免、更正或处理问题和错误的措施。给出的信息应适合于软件版本说明（SVD）预期的受众（例如一个用户机构可能需要避免错误的建议，支持机构则需要改正错误的建议）。



软件开发计划

软件开发计划(Software Development Plan,SDP)描述开发者实施软件开发工作的计划,本文档中“软件开发”一词涵盖了新开发、修改、重用、再工程、维护和由软件产品引起的其他所有活动。软件开发计划是向需求方提供了解和监督软件开发过程、使用方法、每项活动途径、项目安排、组织及资源等的一种手段。

1. 项目概述

1) 工作内容

简要说明项目的各项主要工作,软件的功能、性能等。

2) 产品

说明要交付的软件产品,主要包括程序及文档。

3) 运行环境

运行环境包括硬件环境、软件环境、网络环境等。

4) 最后交付期限

描述移交应交付产品的最后期限。

2. 实施整个软件开发活动的计划

1) 软件开发过程

描述要采用的软件开发过程。计划应覆盖合同中的所有相关条款,确定已计划的开发阶段、目标和各阶段要执行的软件开发活动。

2) 软件开发总体计划

主要书写以下方面的内容:软件开发方法、软件产品标准、可重用的软件产品、处理关键性需求、计算机硬件资源利用、记录原理、需方评审途径。

3. 实施详细软件开发活动的计划

对每项活动的描述应包括以下方面的途径(方法/过程/工具):

(1) 所涉及的分析性任务或其他技术性任务。

(2) 结果的记录。

(3) 与交付有关的准备。

对于下述内容,可有选择地进行描述:项目计划和监督、建立软件开发环境、系统需求分析、系统设计、软件需求分析、软件设计、软件实现和配置项测试、配置项集成和测试、软件

配置项合格性测试、软件配置项/硬件配置项的集成和测试、系统合格性测试、软件使用准备、软件移交准备、软件配置管理、软件产品评估、软件质量保证、问题解决过程、联合评审、文档编制、其他软件开发活动。

4. 进度表和活动网络图

应给出：

(1) 进度表。标识每个开发阶段中的活动，给出每个活动的起始点、提交的草稿、最终结果的可用性、其他里程碑以及每个活动的完成点。

(2) 活动网络图。描述项目活动之间的顺序关系和依赖关系，标识出完成项目中有最严格时间限制的活动。

5. 项目组织和资源

分条描述各阶段要使用的项目组织和资源。

1) 项目组织

描述本项目要采用的组织结构，包括涉及的组织机构以及机构之间的关系、执行所需活动的每个机构的权限和职责。

2) 项目资源

描述适用于本项目的资源。



软件质量保证计划

软件质量保证计划(Software Quality Assure Plan,SQAP)是保证软件质量的重要文档,项目承办单位(或软件开发单位)中负责软件质量保证的机构或个人必须制定。SQAP必须由项目委托单位和项目承办单位(或软件开发单位)的代表共同签字、批准。SQAP规定了在项目中采用的软件质量保证的措施、方法和步骤。

1. 管理

描述负责软件质量保证的机构、任务及其有关的职责。

1) 机构

描述与软件质量保证有关的机构的组成,还必须清楚地描述来自项目委托单位、项目承办单位、软件开发单位或用户中负责软件质量保证的各个成员在机构中的相互关系。

2) 任务

描述计划所涉及的软件生命周期中有关阶段的任务,特别是要把重点放在描述这些阶段所应进行的软件质量保证活动上。

3) 职责

指明软件质量保证计划中规定的每一个任务的负责单位或成员的责任。

2. 文档

列出在该软件的开发、验证与确认以及使用与维护等阶段中需要编制的文档,并描述对文档进行评审与检查的准则。

1) 基本文档

列出需要的基本文档。

2) 用户文档

列出需要的用户文档。

3) 其他文档

列出需要的以上两种文档以外的文档。

3. 评审和检查

规定所要进行的技术和管理两方面的评审和检查工作,并编制或引用有关的评审和检查规程以及通过与否的技术准则。以下评审都需要描述是如何进行评审的,都评审了哪些具体内容。

(1) 软件需求(规格)评审。
(2) 系统/子系统设计评审。
(3) 软件设计评审。
(4) 功能检查。
(5) 物理检查。

4. 评审和审核

1) 过程的评审

描述对项目进行过程评审的方法和依据,列出项目定义的过程以及相应的过程评审。

2) 工作产品的审核

描述进行产品审核的方法和依据。列出在项目过程中应产生的工作产品和质量记录,以及需要由 SQA 负责人审核的工作产品和相应的产品审核活动。

3) 不符合问题的解决

描述过程评审和产品审核的结果怎样形成记录,应形成哪些记录。

描述处理在评审中出现的不符合问题的解决方法。

5. 日程表

列出项目质量保证活动的日程表,并确保质量保证活动的日程表与项目开发计划以及配置管理计划保持一致。



软件配置管理计划

软件配置管理计划(Software Configuration Manager Plan,SCMP)是配置管理的基础性文档,每个软件研发项目都必须制定。SCMP 一般应由项目主管制定,经审核批准后实施。如果软件项目规模较小,或者 SCMP 只是定义一些管理细节且所占篇幅不大,可以将其并入软件开发计划(SDP)或软件质量保证计划(SQAP)之中,作为其中的一部分。

1. 软件配置管理的作用

简述配置管理的作用。

2. 管理

描述负责软件配置管理的机构、任务及其有关的接口控制。

1) 组织结构

给出配置管理的组织结构图。

2) 任务

描述在软件生命周期各个阶段中的配置管理任务以及要进行的评审和检查工作,并指出各个阶段的产品应存放在哪一类软件库中(软件开发库、软件受控库或软件产品库)。

3) 职责

描述与软件配置管理有关的各类机构或成员的职责,并指出这些机构或成员相互之间的关系。

4) 实现

规定实现软件配置管理计划的主要里程碑。

5) 适用的标准、条例和约定

指明所适用的软件配置管理标准、条例和约定,必须说明这些标准、条例和约定需要实现的程度。

描述需要在本项目中编写和实现的软件配置管理标准、条例和约定。

3. 软件配置管理活动

描述配置标识、配置控制、配置状态记录与报告以及配置检查与评审四方面的软件配置管理活动的需求。

1) 配置标识

(1) 详细说明软件项目的基线(即最初批准的配置标识)。

（2）描述本项目所有软件代码和文档的标题、代号、编号以及分类规程。

2）配置控制

（1）描述软件生命周期中各个阶段使用的修改批准权限的级别。

（2）定义对已有配置的修改建议进行处理的方法。

（3）当需要与不属于本软件配置管理计划适用范围的程序和项目进行接口时，本条必须说明对其进行配置控制的方法。

（4）说明与特殊产品（如非交付的软件、现存软件、用户提供的软件和内部支持软件等）有关的配置控制规程。

3）配置状态的记录和报告

（1）指明怎样收集、验证、存储、处理和报告配置项的状态信息。

（2）详细说明要定期提供的报告及其分发办法。

（3）如果有动态查询，要指出所提供的动态查询的能力。

（4）如果要求记录用户说明的特殊状态时，要描述其实现手段。

4. 工具、技术和方法

指明为支持特定项目的软件配置管理所使用的软件工具、技术和方法，指明它们的目的，并在开发者所有权的范围内描述其用法。例如，可以包括用于下列任务的工具、技术和方法：

（1）软件媒体和媒体文档的标识。

（2）把文档和媒体置于软件配置管理的控制之下，并把它正式地交付给用户。例如，要给出对软件库内的源代码和目标代码进行控制的工具、技术和方法的描述；如果用到数据库管理系统，则还要对该系统进行描述。又如，要指明怎样使用软件库工具、技术和方法来处理软件产品的交付。

（3）编制关于程序及其有关文档的修改状态的文档。因此必须进一步定义用于准备多种级别（如项目负责人、配置控制小组、软件配置管理人员和用户等）的管理报告的工具、技术和方法。

实验10 软件用户手册

软件用户手册(Software User Manual,SUM)描述手工操作软件的用户应如何安装和使用一个计算机软件配置项(CSCI)、一组 CSCI、一个软件系统或子系统,还包括软件操作的一些特殊方面,诸如关于特定岗位或任务的指令等。

SUM 是为用户操作软件而开发的,具有要求联机用户输入或解释输出显示的用户界面。如果软件是被嵌入在一个硬件-软件系统中,由于已经有了系统用户手册或操作规程,可能不需要单独的 SUM。

1. 软件综述

1) 软件应用

简要说明软件预期的用途。

2) 软件清单

标识为了使软件运行而必须安装的所有软件文件,包括数据库和数据文件。

3) 软件环境

标识用户安装并运行该软件所需的硬件、软件、手工操作和其他资源。

4) 意外事故以及运行的备用状态和方式

在紧急时刻以及在不同运行状态和方式下用户处理软件的差异。

5) 帮助和问题报告

标识联系方式和应遵循的手续,以便在使用软件遇到问题时获得帮助并报告问题。

2. 访问软件

向用户提供足够的细节,以使用户在学习软件功能细节前能够可靠地访问软件。

1) 软件的首次用户

(1) 熟悉设备。熟悉使用设备的方法。

(2) 访问控制。提供用户可见的软件访问与保密性特点的概述。

(3) 安装和设置。用户在设备上安装、配置、访问、执行、删除软件,覆盖以前的文件或数据,以及输入软件操作的参数必须执行的过程。

2) 启动过程

提供开始工作的步骤,包括任何可用的选项。

3) 停止和挂起工作

描述用户如何停止或中断软件的使用,如何判断是否正常结束或终止。

3. 软件使用指南

向用户提供使用软件的过程。

1）处理过程

解释功能、菜单、屏幕等，描述完成过程必需的次序。

2）相关处理

标识并描述任何关于不被用户直接调用而执行的批处理、脱机处理或后台处理，说明支持这种处理的用户职责。

3）数据备份

描述创建和保留备份数据的过程，这些备份数据在发生错误、缺陷、故障或事故时可以用来代替主要的数据拷贝。

4）错误、故障或紧急情况时的恢复

从处理过程中发生的错误、故障中重启或恢复的详细步骤和保证紧急时刻运行的连续性的详细步骤。

5）消息

完成用户功能时可能发生的所有错误消息、诊断消息和通知性消息，或引用列出这些消息的附录。描述每一条消息的含义和消息出现后要采取的动作。

6）快速引用指南

概述常用的功能键、控制序列、格式、命令或软件使用的其他方面。

参 考 文 献

[1] Barry W. Boehm. 软件工程经济学[M]. 李师贤,等译. 北京:机械工业出版社,2004.

[2] Aye Bakr,Burak Turhan,Aye B. Bener. A New Perspective on Data Homogeneity in Software Cost Estimation[J]. Software Quality Journal,2010. 1.

[3] Christof Ebert, Reiner Dumke. Software Measurement [M]. International Thomson Computer Press,2007.

[4] Janet E. Burge,John M. Carroll,Raymond McCall,et,al. Rationale-Based Software Engineering[M]. Springer Berlin Heidelberg,2008.

[5] Jonas A. Montilva,Beatriz Sandia,Judith Barrios. A Software Engineering Approach[J]. Education and Information Technologies,2002. 3.

[6] Ke Xu,Hèctor Muñoz-Avila. A Case-based Reasoning System For Capturing, Refining and Reusing Project Plans [J]. Knowledge and Information Systems,2008. 2.

[7] M. Ali Babar,Matias Vierimaa,Markku Oivo. Product-Focused Software Process Improvement[M]. USC Center for Software Engineering,2010.

[8] Meir M. Lehman,Juan F. Ramil. Software Evolution and Software Evolution Processes [J]. Annals of Software Engineering,2002. 1.

[9] Mingshu Li,Barry Boehm,Leon J. Osterweil. Unifying the Software Process Spectrum[M]. Springer Berlin / Heidelberg,2005.

[10] Ray Dawson, Bill O'Neill. Simple Metrics for Improving Software Process Performance and Capability [J]. Software Quality Journal,2003. 6.

[11] Valentine Casey. Virtual Software Team Project Management [J]. Journal of the Brazilian Computer Society,2010. 2.

[12] 澳信传媒旗下网站. 配置管理计划编写规范[EB/OL]. http://tech. it168. com /,2009. 4. 12.

[13] 百度空间. 运用界面设计的八条黄金规则[EB/OL]. http://hi. baidu. com/mikeychen/,2009. 11. 3.

[14] 曾映雪. 基于C/S的远程软件维护系统的研究与实现[D]. 河海大学硕士学位论文,2003.

[15] 陈起. 打造软件质量控制体系[J]. 金融电子化,2008. 9.

[16] 陈松乔,任胜兵,王国军. 现代软件工程[M]. 北京:清华大学出版社,2004.

[17] 丁剑洁. 基于度量的软件维护过程管理的研究[D]. 西北大学硕士学位论文,2006.

[18] 硅谷动力. 软件生存周期各个阶段活动定义[EB/OL]. http://www. enet. com. cn,2009. 9. 2.

[19] 郭宁,周晓华. 软件项目管理[M]. 北京:清华大学出版社,2007.

[20] 韩万江,姜立新. 软件项目管理案例教程[M]. 北京:机械工业出版社,2006.

[21] 贺平. 软件测试教程 UML[M]. 北京:电子工业出版社,2005.

[22] 黄叔武,张晓军. 软件项目计划管理[J]. 计算机系统应用,2000. 10.

[23] 蒋晖,王青,李文杰. ISO 9000 在软件组织的实施模型[J]. 计算机工程与设计,2002. 3.

[24] 蒋敏迪. 软件成本估算模型及其实现[D]. 中山大学硕士学位论文,2004.

[25] 金一如,周波,徐斌. 目标驱动的软件质量保证实施[J]. 计算机应用与软件,2007. 4.

[26] 九江学院. 软件编码[EB/OL]. http://www. jju. edu. cn/,2009. 10. 27.

[27] 李德路. 试论软件人员组织与管理[J]. 科技信息,2007. 6.

[28] 李帜,林立新,曹亚波. 功能点分析方法与实践[M]. 北京:清华大学出版社,2005.

[29] 林泉. 浅议软件再工程[J]. 计算机工程与设计,2006.1.
[30] 林子禹,邓万涛,彭德纯,等. 基于面向对象的软件需求分析规范及实施方法研究[J]. 小型微型计算机系统,2009.7.
[31] 毛新军. 面向数据流的软件设计方法[EB/OL]. http://sei.nudt.edu.cn/,2009.9.11.
[32] 梦思月. 软件的可维护性[EB/OL]. http://mengjie213.blog.163.com/blog/,2010.3.31.
[33] 丘丽琴. 软件质量保证的实践与总结[J]. 科技管理研究,2004.4.
[34] 任永昌,邢涛,鄂旭. 软件项目开发过程管理[M]. 北京:北京交通大学出版社,2010.
[35] 任永昌,邢涛,王立,赵宝永. 软件项目管理[M]. 长春:吉林大学出版社,2011.
[36] 任永昌. 基于挣值管理的软件开发成本控制方法的研究[J]. 中国管理信息化. 2007.1.
[37] 任永昌. 软件项目开发方法与管理[M]. 北京:清华大学出版社,2011.
[38] 石柱. 软件工程标准手册[M]. 北京:中国标准出版社,2004.
[39] 史永辉,刘曲明,宋艳芳. 软件需求分析的进一步研究[J]. 情报指挥控制系统与仿真技术,2003.4.
[40] 孙大伟. 我国软件企业人才资源管理问题探讨[D]. 北京交通大学工程硕士学位论文,2003.
[41] 覃征,杨利英,高勇民,等. 软件项目管理[M]. 北京:清华大学出版社,2004.
[42] 王强,曹汉平,贾素玲,木林森. IT 软件项目管理[M]. 北京:清华大学出版社,2004.
[43] 王如龙,邓子云,罗铁清. IT 项目管理——从理论到实践[M]. 北京:清华大学出版社,2008.
[44] 王先国. 软件工程实践教程[M]. 北京:电子工业出版社,2010.
[45] 王应明. 技术经济学[M]. 北京:中国经济出版社,1998.
[46] 王勇,方志达. 项目可行性研究与评估[M]. 北京:中国建筑工业出版社,2004.
[47] 卫红春. 信息系统分析与设计[M]. 西安:西安电子科技大学出版社,2003.
[48] 魏国荣. 核心通信软件维护过程研究及工具实现[D]. 北京邮电大学硕士学位论文,2007.
[49] 肖刚,古辉,程振波,等. 实用软件文档写作[M]. 北京:清华大学出版社,2005.
[50] 薛峰. 软件配置管理中变更管理技术的研究[D]. 大连海事大学工程硕士学位论文,2006.
[51] 薛四新,贾郭军. 软件项目管理[M]. 北京:机械工业出版社,2004.
[52] 阳王东,曾强聪,吴宏斌. 软件项目管理方法与实践[M]. 北京:中国水利水电出版社,2009.
[53] 杨理琴. 软件企业质量管理体系有效性评价与对策研究[D]. 成都理工大学硕士学位论文,2008.
[54] 余忠,李秀珠. 软件企业人力资源开发与管理的策略[J]. 闽江学院学报,2004.10.
[55] 张保军. 软件项目研发团队建设探析[J]. 中国金融电脑,2007.12.
[56] 张海藩. 软件工程导论. 5 版[M]. 北京:清华大学出版社,2008.
[57] 张海藩. 软件工程导论. 4 版[M]. 北京:清华大学出版社,2003.
[58] 张华. 软件项目管理原则谈[EB/OL]. http://www.cdd.cn,2009.7.5.
[59] 张家浩. 软件项目管理[M]. 北京:机械工业出版社,2005.
[60] 张建成,田青,李刚,等. 软件工程需求分析方法探讨[J]. 信息技术与信息化,2007.6.
[61] 郑人杰,殷人昆,陶永雷. 实用软件工程. 2 版[M]. 北京:清华大学出版社,1997.
[62] 中国数字图片网. 软件项目管理原则谈[EB/OL]. http://www.cdd.cn/homepage,2009.7.6.
[63] 朱少民. 软件质量保证和管理[M]. 北京:清华大学出版社,2007.